

MSX

日立パーソナルコンピュータ

H1

かためい
形名

MB-H1 / MB-H1E

とり あつかい せつ めい しょ
取扱説明書



この写真は、MB-H1です。

 HITACHI

MB-H1とMB-H1Eの違い

この取扱説明書はH1シリーズのMB-H1とMB-H1E共通の
説明書となっています。MB-H1とMB-H1Eの違いは次のと
おりです。

1、RAM容量

MB-H1 : 32Kバイト

MB-H1E : 16Kバイト

2、メニュー画面

MB-H1のみ装備しています。

3、インターフェース

MB-H1EはRF出力はありません。

したがって、MB-H1Eはビデオ入力端子のないテレビに接続す
ることはできません。

また、MB-H1Eで市販のMSX-BASICのソフトウェア
(ゲームカートリッジなど)を使用する場合には、16Kバイトを越
える記憶容量を必要とするソフトウェアは、使用できません。

MB-H1のみ マークについて

この取扱説明書はH1シリーズのMB-H1とMB-H1E
共通の説明書となっています。

MB-H1にのみある機能などを説明しているところには、

MB-H1のみ マークをつけてあります。

このたびは日立パーソナルコンピュータをお求めいただき、まことにありがとうございました。この「取扱説明書」をよくお読みいただき、正しくご使用ください。なお、お読みになった後は保証書、ご相談窓口一覧表とともに大切に保存してください。

H1 取扱説明書の使い方

この取扱説明書は全体が3つの部分からできています。

取扱編

H1を使うときに気をつけなければいけないことや、H1とテレビやカセットテープなどの接続のしかた、キーボードの正しい扱い方、電源の入れ方など、操作の方法や取り扱い方を説明します。

ベーシック基礎編

自分でプログラムを作ったり、雑誌などにのっているプログラムを改良したりするときに、必要となるベーシックの基本的な命令や文法をわかりやすく説明します。

ベーシック文法編

本格的に、ベーシックでプログラムを作ったり、より高度にH1を使うときに必要となるコマンドやステートメント、関数を説明します。使うときに便利のように、コマンド、ステートメント、関数は機能別に分けてあります。

取扱編は、これからH1を使っていくのに必要なことばかりですから、今までにコンピュータを使ったことのある人でも、必ず一度は読むようにしてください。

ベーシック基礎編は、初めてコンピュータを使う人は必ず読んでください。

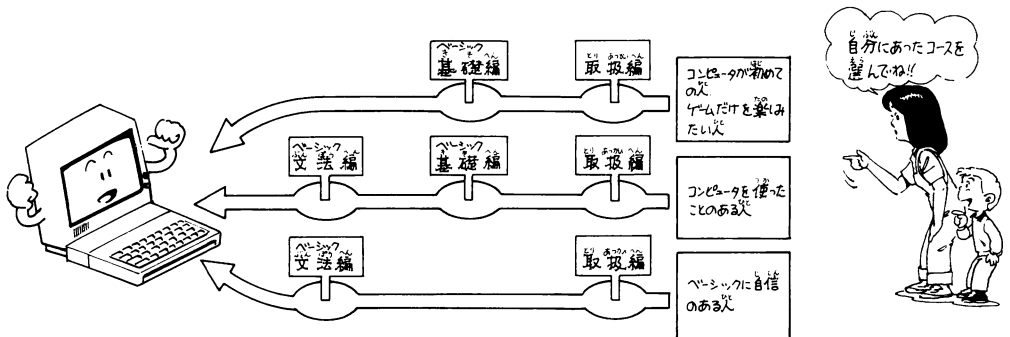
実際にプログラムが作れるようになりますよ。

ゲームを楽しみたいという人も、できるだけここまで読んでください。ゲームがもっとおもしろくなりますよ。

コンピュータを使ったことのある人は、復習のつもりで読んでください。

ベーシックに自信のある人は、「ベーシック基礎編」とばして、次の「ベーシック文法編」にすすんでもかまいません。

ベーシック文法編は、かなりむづかしい内容もありますので、H1で初めてコンピュータを使う人は、「ベーシック基礎編」をよく理解してからがよいでしょう。



● ご注意 ●

1. 本書の内容の一部または全部を無断で転載することは、禁止されています。
2. 本書の内容について、予告なしに変更することがあります。
3. 本書は内容について、万全を期しておりますが、万一わからない点や誤り、お気付きの点がありましたらご一報くださいますようお願い致します。
4. 運用した結果の影響については、3項にかかわらず責任を負いかねますのでご注意ください。
5. 本書に記載した写真、イラストなどは、実際の外観とは一部異なる場合がありますのでご了承ください。

とり あつかい へん もく じ 取 扱 編 目 次

I	H1 ^{し しょう} を使用する ^{まえ} 前に	
1.	部品 ^{ぶひん} を ^た 確かめよう	15
2.	これだけは ^き 気をつけよう	16
II	H1 ^{し しょう} の ^{かい} ご紹介	
1.	主な ^{おも} 特長 ^{とくちやう}	21
2.	H1 のシステム	22
III	どうすれば ^{うご} 動くの ^{せつぞく} （ ^く 接続と ^な 組み立て）	
1.	各部 ^{かくぶ} の ^{めいしやう} 名称とはたらき	24
2.	接続 ^{せつぞく} のしかた	27
IV	さあ ^{うご} 動かしてみよう	
1.	電源 ^{でんげん} の ^い 入れ方 ^{かた}	34
2.	H1 が ^{うご} 動きはじめる	35
3.	カートリッジの ^{あつか} 扱い ^{かた} 方	38
4.	キーボードの ^{あつか} 扱い ^{かた} 方	39
V	MB-H1 のメニュー ^{が めん} 画面 ^{つか} の ^{かた} 使い方	54
VI	おや？ H1 が ^{こ しょう} 故障 ^{おも} かなと思ったら	62
VII	H1 の ^{し しょう} 仕様	64
VIII	アフターサービスと ^{ほ しょう} 保証	65

ベーシック基礎編 目次

I とにかく使ってみよう

ベーシックって何?…まずはじめに	70
画面に何か書いてみよう…一番やさしいPRINT命令	71
計算だってできるんだ…たし算、ひき算、かけ算、わり算	74

II プログラムの基礎

プログラムって何?…プログラムを作る前に(CLS)	78
コンピュータは変数が大好き…変数、代入文の使い方	79
行番号で順序よく…行番号の考え方(NEW, END, RUN)	82
プログラムを書き換えよう…プログラムの修正(LIST)	84
プログラムを保存するには…セーブ、ロードの方法(CSAVE, CLOAD, CLOAD?)	87
プリンタを使おう(LPRINT, LLIST)	91

III ベーシックを勉強しよう

はじめに(AUTO, RENUM, DELETE, REM)	93
GOTO文で流れをかせよう…GOTOの使い方(GOTO, CONT)	96
「,」と「;」の大研究…さらにくわしいPRINT命令	98
PRINTを助ける三銃士…PRINTといっしょに使う便利な命令や関数 (SPC関数、TAB関数、LOCATE)	100
くり返しに便利な命令…FOR~NEXTの使い方	103
文字列って何?…文字列の使い方	107
文字列料理の3大道具…文字列を抜き出そう (LEFT\$関数、RIGHT\$関数、MID\$関数、LEN関数)	109
こちらINPUT応答せよ	112
数の料理の専門家…数値関数(ABS関数、SGN関数、INT関数)	116

でたらめな数をRND関数で作ろう	119
「:」を使ってプログラムを短く……マルチ ステートメントの使い方	121
判断はIF～THENにおまかせ(IF～THEN～ELSE)	122
ゲームをするならINKEY\$関数が便利	125
デバッグのお話	127
COLORで色をつけよう…カラー画面の作り方	131
お絵かきをする前に(SCREEN)	133
H 1で絵を描こう(PSET, STEP, PRESET)	135
直線や長方形も簡単に(LINE, PAINT)	138
CIRCLEで円も描けるぞ	142
スプライトで図形を動かそう (SPRITE\$, PUT SPRITE)	144
文字と数字を交換しよう…VAL関数、STR\$関数の使い方	151
キャラクターコードのお話(CHR\$関数、ASC関数)	154
H 1を時計にしよう…時計機能とは(TIME)	157
H 1は音だって出せるんだ(BEEP, PLAY)	159
同じような処理にはサブルーチンを使おう!…GOSUB～RETURNの使い方	163
配列って何?(DIM)	165
READ～DATAでデータを使おう(READ, DATA, RESTORE)	171
とっても便利なON～GOTO, ON～GOSUB	174
関数を作ってみよう(DEF FN)	176
H 1の中身をのぞいてみよう(PEEK関数、POKE、CLEAR)	178

ぶんぼうへん もくじ ベーシック文法編 目次

はじめに	183
------------	-----

MSX-BASICの概要 <small>がいよう</small>	184
--	-----

コマンド

LIST(リスト)	189
LLIST(エル リスト)	190
AUTO(オート)	190
RENUM(リナンバー)	191
DELETE(デリート)	191
NEW(ニュー)	192
RUN(ラン)	192
CONT(コンティニュー)	193
TRON/TROFF(トレース オン/トレース オフ)	193
CSAVE(シーセーブ)	194
CLOAD(シーロード)	194
CLOAD?(シーロードベリファイ)	195
LOAD(ロード)	195
SAVE(セーブ)	196
BLOAD(ビーロード)	196
BSAVE(ビーセーブ)	196
MERGE(マージ)	197
CLEAR(クリア)	198

いっばん 一般ステートメント

LET(レット)	199
REM(リマーク)	200
FOR~NEXT(フォー~ネクスト)	200
GOSUB~RETURN(ゴースブ~リターン)	201

GOTO(ゴートウー)	201
IF~THEN~ELSE(イフ~ゼン~エルス)	202
ON GOTO / GOSUB(オン ゴートウー/ ゴーサブ)	203
STOP(ストップ)	203
END(エンド)	204
INPUT(インプット)	204
LINE INPUT(ライン インプット)	205
PRINT(プリント)	205
PRINT USING(プリント ユージング)	206
LPRINT(エル プリント)	207
LPRINT USING(エルプリント ユージング)	208
READ(リード)	208
DATA(データ)	209
RESTORE(リストア)	209
DIM(ディメンジョン)	210
ERASE(イレース)	210
DEF FN(デファイン ファンクション)	211
DEF INT/SNG/DBL/STR(デファイン インテジャー /シングル/ダブル/ストリング)	211
MID\$(ミドル ドル)	212
SWAP(スワップ)	213

にゅうしゅつりょく

入出力ステートメント

MAXFILES(マックスファイルズ)	214
OPEN(オープン)	215
CLOSE(クローズ)	215
PRINT # (プリント シャープ)	216
PRINT # USING(プリント シャープ ユージング)	216
INPUT # (インプット シャープ)	217
LINE INPUT # (ライン インプット シャープ)	218
INPUT\$(インプット ドル)	219

がめん

画面・グラフィック ステートメント

CLS(クリア スクリーン)	220
LOCATE(ロケイト)	220
WIDTH(ウイドス)	221

COLOR(カラー).....	221
SCREEN(スクリーン).....	222
PSET(ピーセット).....	223
PRESET(ピーリセット).....	224
LINE(ライン).....	224
CIRCLE(サークル).....	225
PAINT(ペイント).....	225
PUT SPRITE(プット スプライト).....	226
DRAW(ドロー).....	227

おんがくえんそう

音楽演奏ステートメント

BEEP(ビーブ).....	228
PLAY(プレイ).....	228
SOUND(サウンド).....	230

かんけい

ファンクションキー関係ステートメント

KEY(キー).....	233
KEY LIST(キー リスト).....	234
KEY ON/OFF(キー オン/オフ).....	234

しゅり

エラー処理ステートメント

ERROR(エラー).....	235
ON ERROR GOTO(オン エラー ゴートウー).....	236
RESUME(リジューム).....	236

わこしゅり

割り込み処理ステートメント

ON INTERVAL GOSUB(オン インターバル ゴーサブ).....	238
INTERVAL ON/OFF/STOP(インターバル オン/オフ/ストップ).....	239
ON KEY GOSUB(オン キー ゴーサブ).....	239
KEY ON/OFF/STOP(キー オン/オフ/ストップ).....	240
ON SPRITE GOSUB(オン スプライト ゴーサブ).....	241
SPRITE ON/OFF/STOP(スプライト オン/オフ/ストップ).....	241
ON STOP GOSUB(オン ストップ ゴーサブ).....	242
STOP ON/OFF/STOP(ストップ オン/オフ/ストップ).....	243
ON STRIG GOSUB(オン エストリガ ゴーサブ).....	243
STRIG ON/OFF/STOP(エストリガ オン/オフ/ストップ).....	244

とく しゅ

特殊ステートメント

POKE(ポーク)	245
VPOKE(バイポーク)	246
DEF USR(デファイン ユーザ)	246
OUT(アウト)	247
WAIT(ウェイト)	247
CALL(コール)	248
MOTOR ON/OFF (モータ オン/オフ)	248

すう ち かんすう

数値関数

ABS(アブソリュート)	249
FIX(フィックス)	249
INT(インテジャー)	250
RND(ランダム)	250
SGN(サイン)	251
SIN(サイン)	251
COS(コサイン)	252
TAN(タンジェント)	252
ATN(アークタンジェント)	252
SQR(スクエア ルート)	253
EXP(エクスポネンシャル)	253
LOG(ログ)	254
CINT(シー インテジャー)	254
CDBL(シー ダブル)	255
CSNG(シー シングル)	255

も じ かんすう

文字関数

LEFT\$(レフト ドル)	256
RIGHT\$(ライト ドル)	257
MID\$(ミドル ドル)	257
LEN(レングス)	258
ASC(アスキー)	258
CHR\$(キャラクター ドル)	258
VAL(ハル)	259
STR\$(ストリング ドル)	259
STRING\$(ストリング ドル)	260

SPACE\$(スペース ドル)	260
INSTR(インストリング)	261
INKEY\$(インキー ドル)	261
INPUT\$(インプット ドル)	262
BIN\$(バイナリー ドル)	262
OCT\$(オクト ドル)	263
HEX\$(ヘキサ ドル)	263

とく しゅ かん すう 特殊関数

PEEK(ピーク)	264
VPEEK(バイピーク)	264
USR(ユーザ)	265
VARPTR(バーポインター)	266
INP(インプット)	266
TAB(タブ)	267
SPC(スペース)	267
FRE(フリー)	268
EOF(エンド オブ ファイル)	268
ERL(エラー ライン)	269
ERR(エラー)	269
CSRLIN(カーソル ライン)	270
LPOS(エル ポジション)	271
PAD(パッド)	271
PDL(パドル)	272
PLAY(プレイ)	272
POINT(ポイント)	273
POS(ポジション)	273
STICK(スティック)	274
STRIG(エストリガ)	274

とく しゅ へん すう 特殊変数

TIME(タイム)	275
SPRITE\$(スプライト ドル)	276
VDP(バイデーピー)	276
BASE(ベース)	277

索引 (コマンド ステートメント アルファベット順)	279
-----------------------------------	-----

索引 (関数・特殊変数 アルファベット順)	282
------------------------------	-----

付 録 目 次

サンプルプログラム	287
MB-H1のモニタ機能仕様	297
VDPについて	305
アドレスマップ	310
コントロールコード一覧表	311
キャラクターコード一覧表/ カラーコード一覧表	312
参考図書	313
エラーメッセージ一覧表	314

とりあつかいへん 取扱編

H1でゲームをしたり、プログラムを作るときに、まず知っていなければならないのが、H1の正しい扱い方です。

H1は、とても精密な機械ですから、大切に取り扱いってください。この取扱編では、H1を使うときに気をつけなければならないことや接続のしかた、キーボードの扱い方など、正しくH1を扱うために必要なことを説明します。

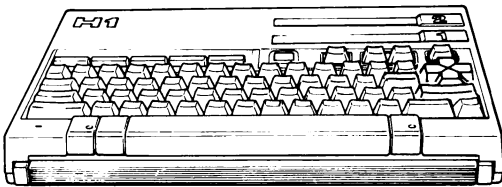
I H1 ^{し よう}を使用する^{まえ}前に

1. 部品^{ぶ ぴん}を^{たし}確かめよう!

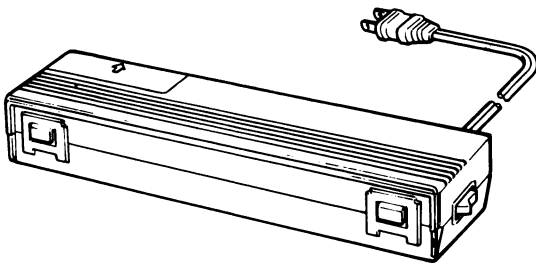
H1^{はい}が入^{はい}っていた箱^{はこ}の中^{なか}には、次^{つぎ}のものがいっしょに入^{はい}っています。

確^{たし}かめてみましょう。

(1). 本^{ほん}体^{たい}

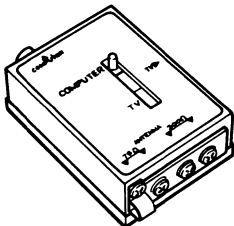


(2). 電^{でん}源^{げん}ボック^すス

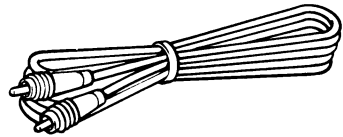


(3). スイ^しッチ^{ごう}ボック^{きりかえ}ス(テレビ信号切^{しんごうきりかえ}換^{よう}用)

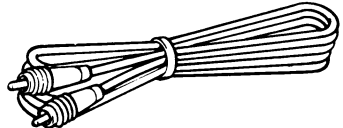
MB-H1のみ



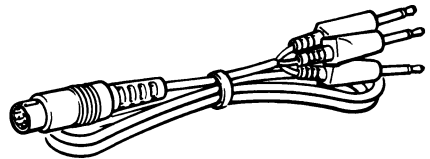
(4). テレ^{せつ}ビ^{ぞく}接^{せつ}続^{ぞく}ケー^ぶブル



(5). オー^{せつ}ディ^{ぞく}オ接^{せつ}続^{ぞく}ケー^ぶブル(MB-H1Eのみ)

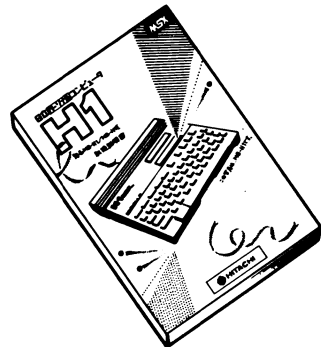


(6). カセ^{せつ}ットレ^{ぞく}コー^{せつ}ダ接^{せつ}続^{ぞく}ケー^ぶブル



(7). 保^ほ証^{しう}書^{しょ} (箱^{はこ}に貼^はってあ^りま^す。)

(8). 取^{とり}扱^{あつか}説^{せつ}明^{めい}書^{しょ}



(9). グラ^もフィ^じック文^{ぶん}字^じシ^しール

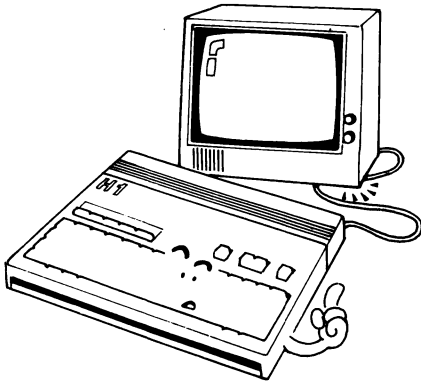
(10). ファン^{ひやう}クシ^じョンキー表^{ひやう}示^じシ^しート

2. これだけは気をつけよう

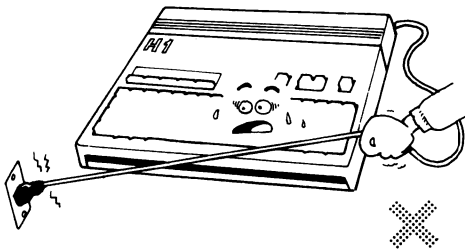
H1が間違っ^{まちが}た動^{どう}作^{さく}をし^したり、こわ^これてしま^{しま}わ^わないよ^ように、次^{つぎ}のこ^ことに注^{ちゅう}意^いしてくだ^{くだ}さい。

■ 電^{でん}源^{げん}コ^こー^ど

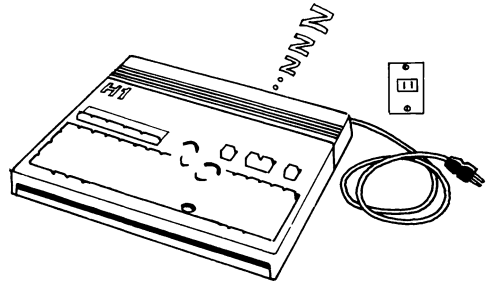
- 電^{でん}源^{げん}コ^こー^どの^{うへ}にH1やテレ^おビ^びを置^おいた^りして傷^{きず}をつ^つけ^けないよ^ようにしま^{しま}し^しよう。
(電^{でん}源^{げん}コ^こー^どに傷^{きず}がつ^ついた^たま^ま、H1をつ^{つか}う^うのはと^とて^とも危^き険^{けん}です。)



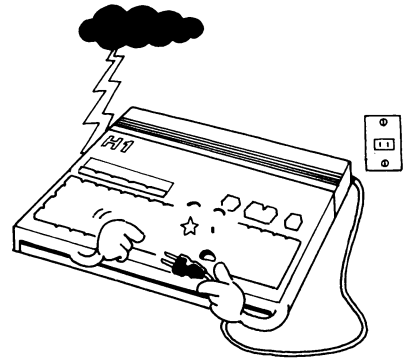
- コンセ^こント^んか^から^ら抜^ひく^くと^ときは、プ^ぷラ^らグ^ぐを持^もっ^つて^て抜^ひき^きま^まし^しよう。
(電^{でん}源^{げん}コ^こー^どを持^もっ^つて^て抜^ひく^くとコ^こー^どが^がい^いた^たみ^みま^ます。)



- 使^{つか}わ^わない^いと^ときは電^{でん}源^{げん}コ^こー^どを^をコ^こン^こセ^せント^{んと}か^から^ら抜^ひい^いて^てお^おき^きま^まし^しよう。

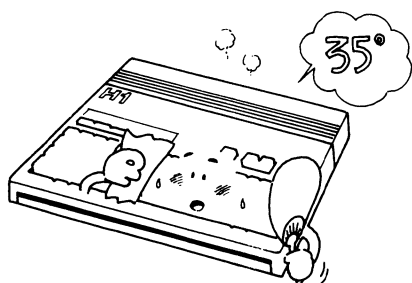


- 雷^{かみなり}が^がな^なった^たら^ら、H1の^きス^いッ^ちを^を切^きり^り電^{でん}源^{げん}コ^こー^どを^をコ^こン^こセ^せント^{んと}か^から^ら抜^ひい^いて^てお^おき^きま^まし^しよう。

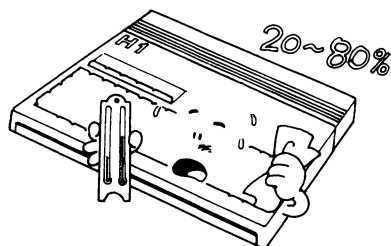


■ H1 を使う場所

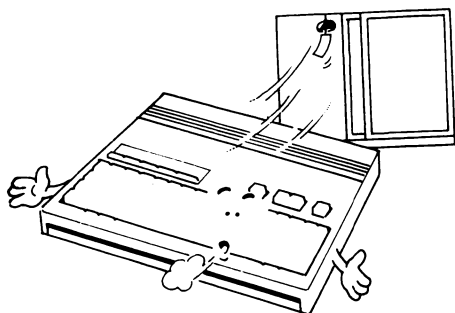
- 室温は0℃～35℃の場所で使いましょう。それ以上暑かったり寒かったりすると間違った動作をすることがあります。特に直射日光の当たる場所、ストーブのような熱器具の近くなど高温となる場所で使うのはやめましょう。



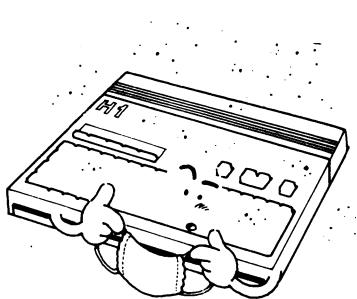
- 湿度は20%～80%の場所で使いましょう。



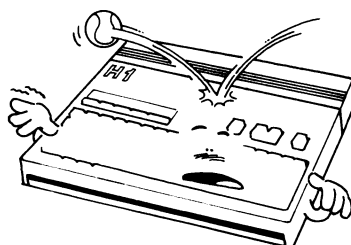
- 風通しのよい場所で使いましょう。また、H1の通風孔をふさいではいけません。



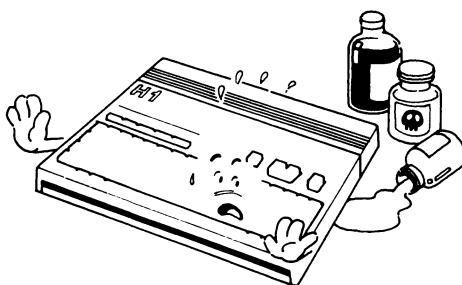
- ほこりの少ない場所で使いましょう。



- 振動のある場所で使うのはやめましょう。またぶつけたり、落としたり、乱暴な扱い方をすると故障の原因になります。

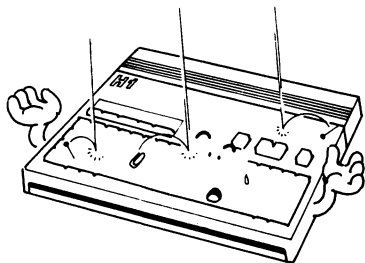


- 薬品を使う場所や薬品に触れるような場所で使ったり、保管するのはやめましょう。また、殺虫剤がかからないようにしてください。

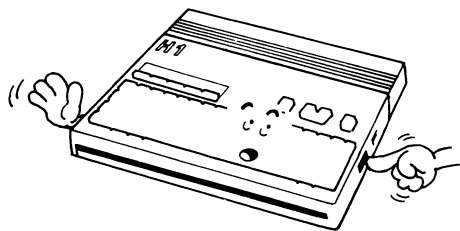


■ H1の取り扱いについて

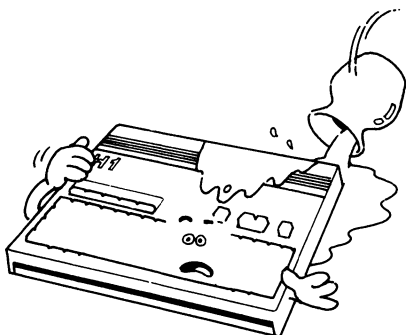
- H1のすきま(特にカートリッジスロット)などから、ピンやクリップなどの金属や紙などを差し込んだり、落としたりしないようにしましょう。故障の原因になります。



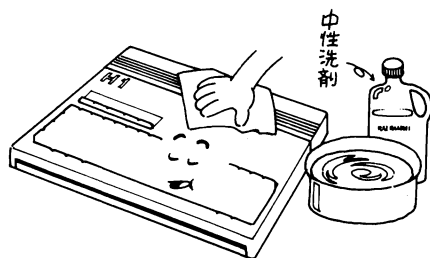
- コネクタの金属の部分に手を触れないようにしましょう。間違った動作をすることがあります。



- 間違っ^{まちが}て H1の中^{なか}に水^{みず}などが入^{はい}ったときは、すぐに電源コードのプラグをコンセントから抜いて H1を買^かった店^{みせ}に相^{そう}談^{だん}しましょう。

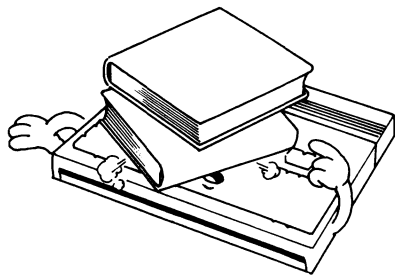


- H1を手入れするときは、やわらかい布でからぶきしてください。よごれのひどいときには、中性洗剤をうすめて、布にひたして箇くしほり、汚れをふきとってください。



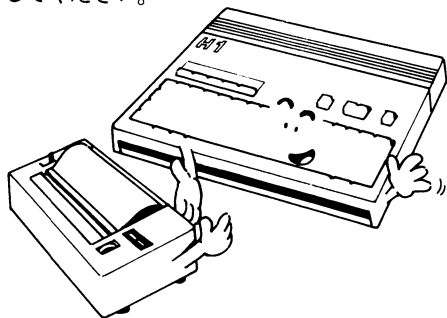
(ベンジン、シンナー、化学ぞうきんなどを使^{つか}ってはいけません。)

- H1の上^{うへ}に物^{もの}を置^おかないようにしましょう。

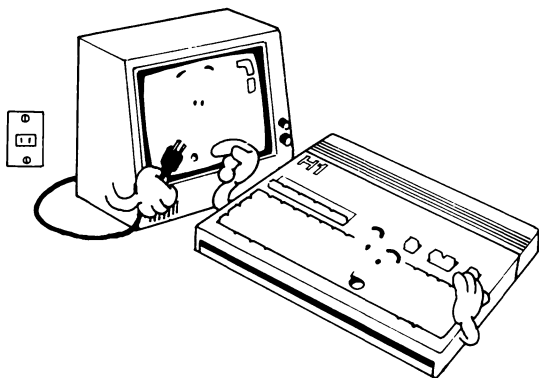


■その他

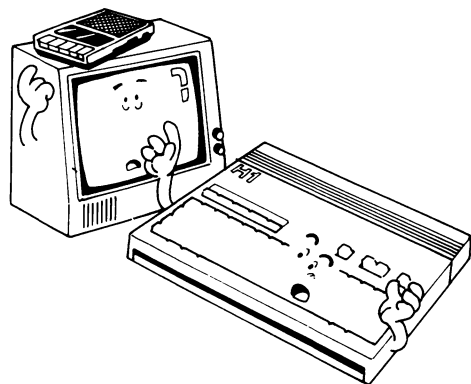
- H1に接続する装置は、必ず指定されたものを使いましょう。それ以外の装置では、正しく動かないことがあります。また、指定された装置を使うときには、その装置の取扱説明書のとおりにお操作してください。



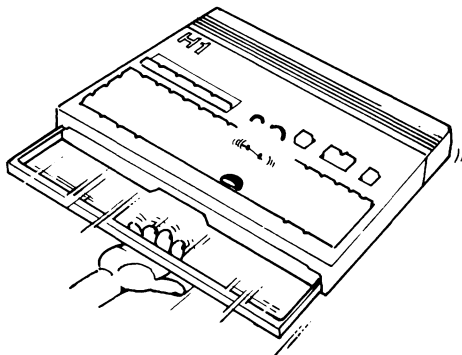
- H1を使用しているときには、テレビの電源を切つてはいけません。間違つた動作をすることがあります。



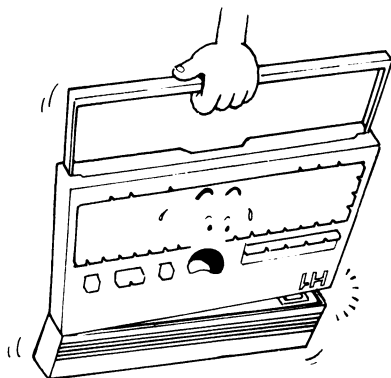
- カセットレコーダはテレビのすぐ近くに置かないようにしましょう。またH1をテレビの上で使用しないようにしましょう。



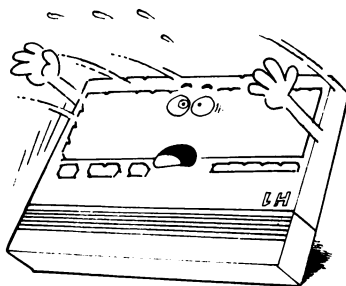
- ハンドルをひき出して使うとき無理な力を加えないようにしましょう。



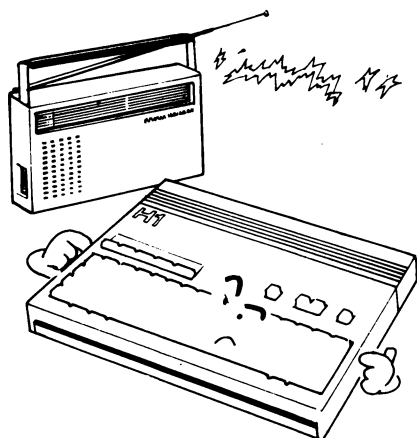
- H1を持ち運ぶとき、電源ボックスをきちんと固定するようにしましょう。詳しくは、27ページの接続のしかたをご覧ください。



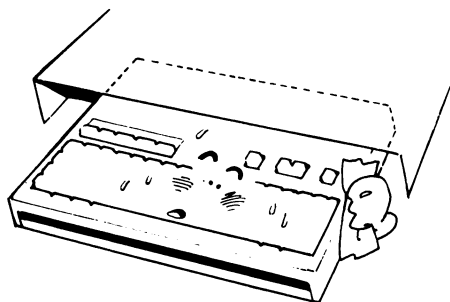
- セットを立てて置かないようにしましょう。倒れると故障の原因となります。



- ラジオやテレビなど近くで使用しますと、ラジオやテレビに雑音が入ることがあります。また強い磁界や雑音を発生する装置などが近くにありますと、逆にH1に雑音が入り正常に動作しない場合があります。このような場合は、離してご使用ください。



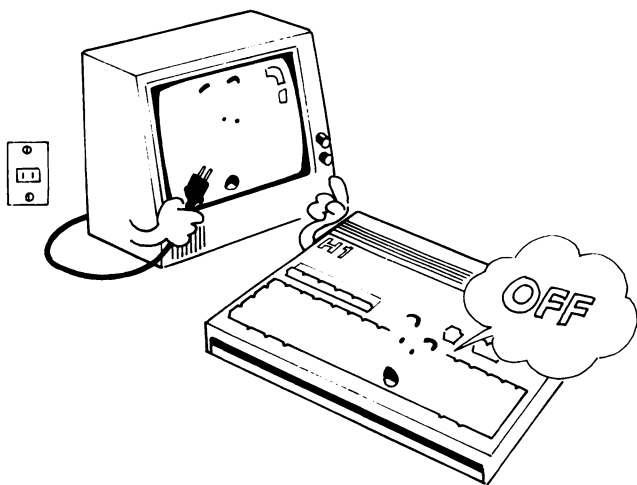
- 狭い場所に押し込んで、長時間H1を使用すると電源ボックスが熱くなることがありますので取り扱いには注意してください。



■もし、H1の様子がおかしいときは

- 万一、故障したり異常に気がついたら、すぐ電源スイッチを切り電源プラグをコンセントから抜いて、H1を買ったお店に修理を依頼してください。

- 周辺機器を接続するときは、必ずH1と周辺機器の電源を切りましょう。



II H1のご紹介

1. 主な特長

● 本体はほぼA4サイズ、重さ1.8kg *

(MB-H1Eは1.7kg) *

大判の大学ノート(A4サイズ)とほぼ同じでしかも、1.8kg (MB-H1Eは1.7kg) という軽さですから、楽に持ち運びができます。友達の家に持って行ってH1で遊ぶのも簡単です。

* 電源ボックスは除く

● 初めての人も操作が簡単なメニュー画面

MB-H1のメニュー画面

画面に表示されたキーを選んで押すだけで、BASIC、モニタ、ROM、スケッチ、サウンドの5つの機能を楽しむことができます。

● MSX-BASIC

H1はMSX-BASICを標準実装しています。もともとビジネス用のベーシックですから、機能は豊富です。本格的にベーシックを学ぼうという方にも、十分に満足していただけます。

● 家にあるテレビに接続するだけ

専用のディスプレイ(表示装置)は必要ありません。皆さんの家にあるテレビがディスプレイです。接続も簡単です。

注) MB-H1Eはビデオ入力端子を持つテレビのみに接続できます。

● スピードコントロール

スピードコントロールスイッチがついていますから、ゲームを3段階のスピードで楽しむことができます。

● 使いやすく、疲れの少ないキーボード

キーボードのキーには、少しずつ角度がついています(ステップスカルプチャータイプ)。そのため、使い心地がよく、疲れが少なくなりました。これで長～いプログラムも大丈夫。また、かなキーの配列は初心者でも使いやすいように、50音順(あいうえお順)になっています。

● 楽しいグラフィックとサウンド機能

美しい16色の画像を表示することができ、しかも文字や図形をアニメーションのように動かすことのできるスプライト機能も持っています。音楽の好きな人には8オクターブ3重和音のサウンド機能が用意されています。

● いろいろなものがすぐに使えます

H1はジョイスティック、カセットテープレコーダ、プリンタのインターフェースを内蔵していますから、ジョイスティックでゲームを楽しんだり、プログラムを保存したり、プリンタで打出したりすることも簡単です。

MSXとは……

CPUにZ-80A、システムソフトウェアにマイクロソフト社製のMSX-BASICを用いた、パーソナルコンピュータです。

MSX仕様のパソコンには**MSX**マークが表示される**MSX**マークのROMカートリッジやテープで提供される応用ソフトウェアが共通して使えます。

MSXマークはマイクロソフトの商標です。

2. H1のシステム

(H1の仲間達)

●周辺装置

H1は本体だけでは、何もすることができません。考えたり、計算したりすることはできるのですが、その答えを私たちに伝えるものがないからです。そこで、H1にテレビを接続して画面で答えを伝えさせるようにするのです。このように、私たち人間にH1からの言葉を伝える装置を、出力装置といいます。

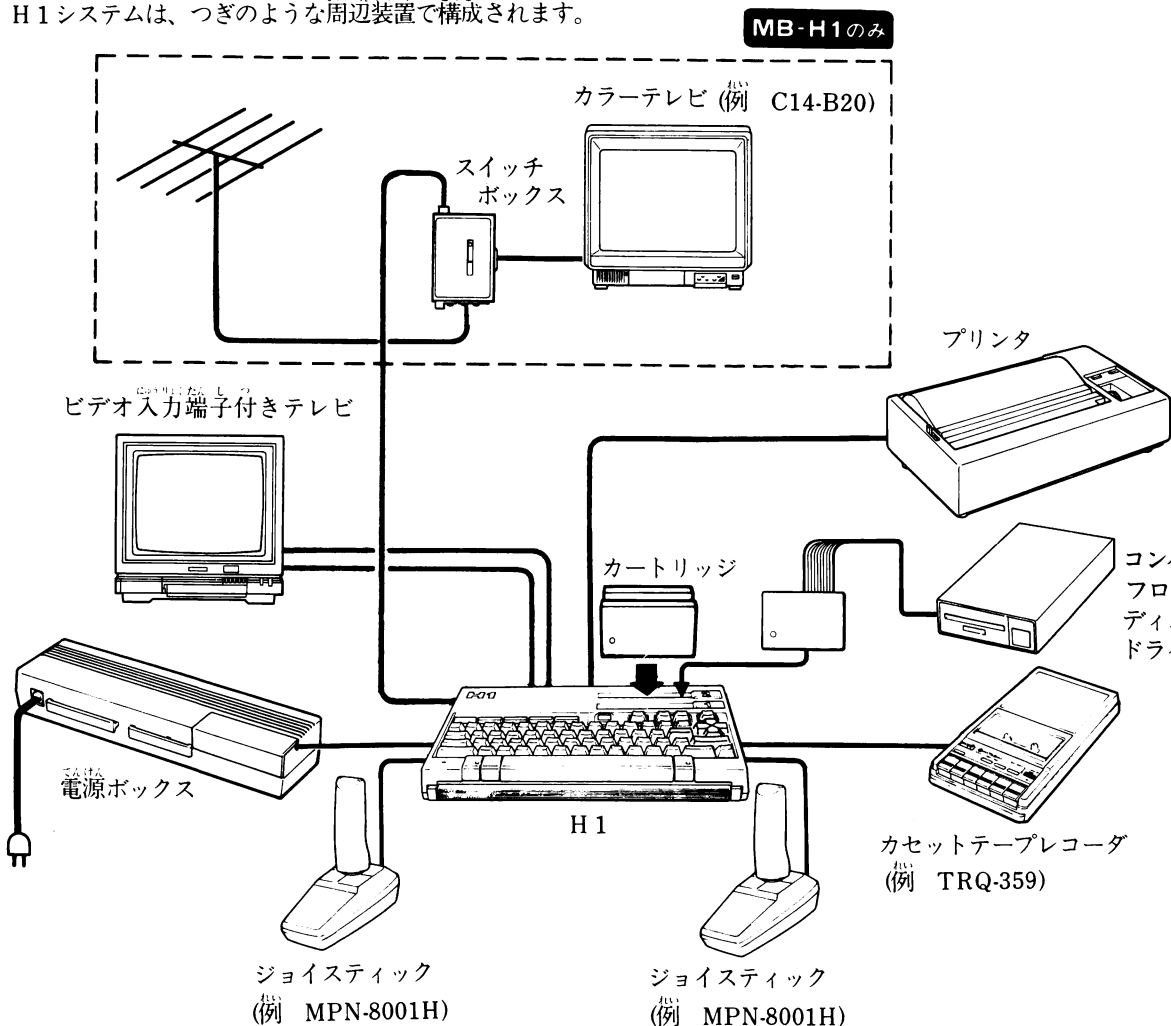
出力装置にはこの他にプリンタがあります。H1からのことばを、紙に打ち出して伝えます。

H1にはキーボードがあります。キーボードは、私たち人間がH1に言葉や命令などを伝えるためのものです。このような装置を入力装置といいます。ジョイスティックも入力装置です。この他にも、カセットテープレコーダやコンパクトフロッピーディスクドライブのように、H1からのことばを記憶したり、記憶したことばをH1に伝えたりして、入力装置と出力装置の両方のはたらきをする入出力装置というものもあります。

このような装置をまとめて、周辺装置といいます。また、H1とその周辺装置をまとめてシステムといいます。

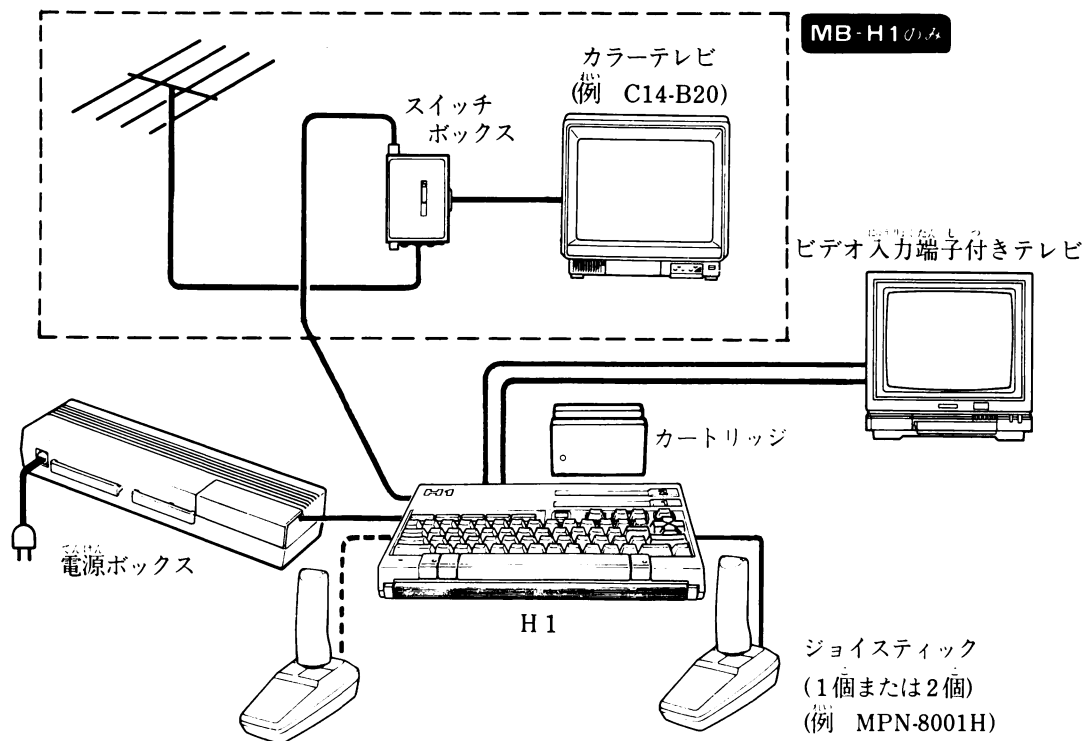
●H1システムアップ構成

H1システムは、つぎのような周辺装置で構成されます。



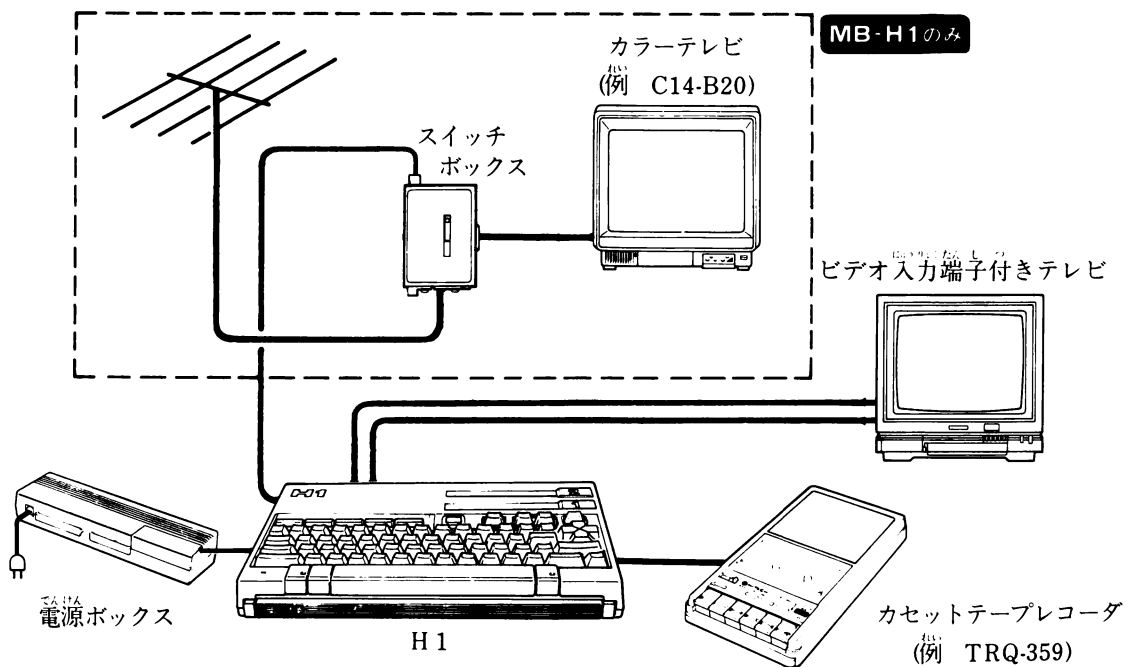
●ゲーム中心のシステム

H1 + カラーテレビ + ジョイスティック 1 個または 2 個 + ゲームカートリッジ



●ベーシック中心のシステム

H1 + カラーテレビ + カセットテープレコーダ

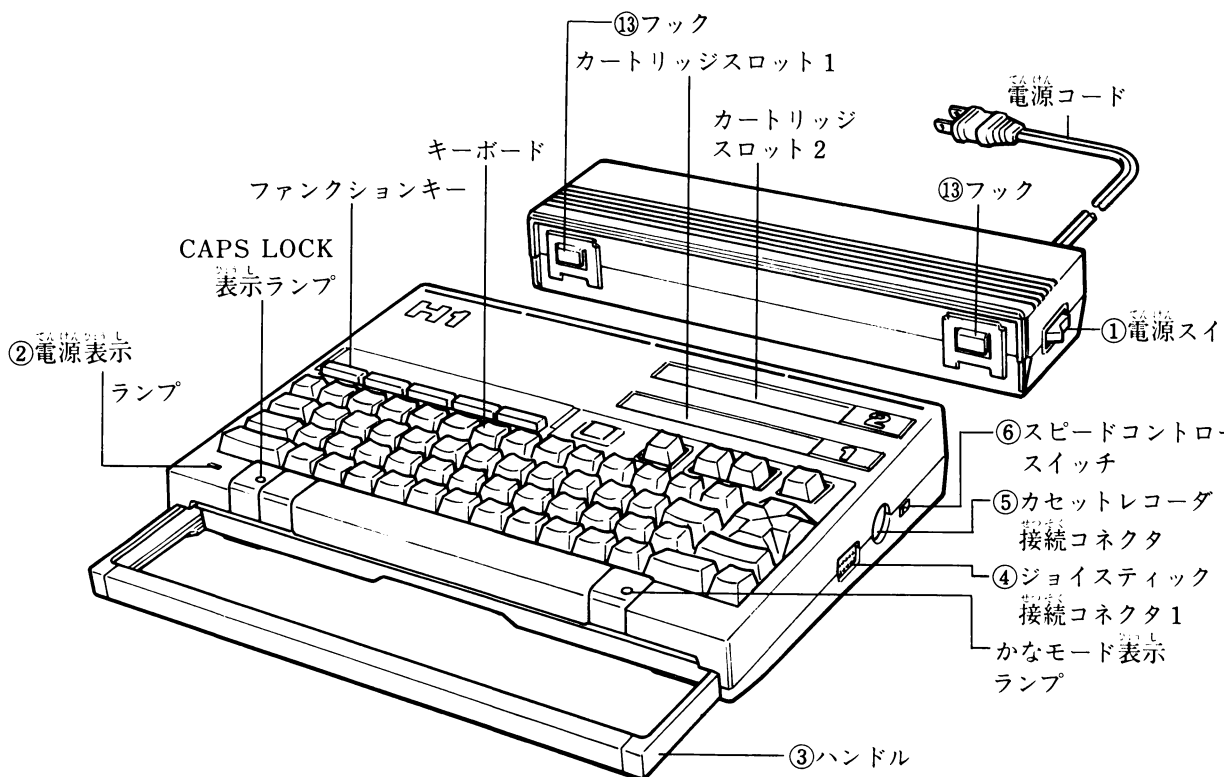


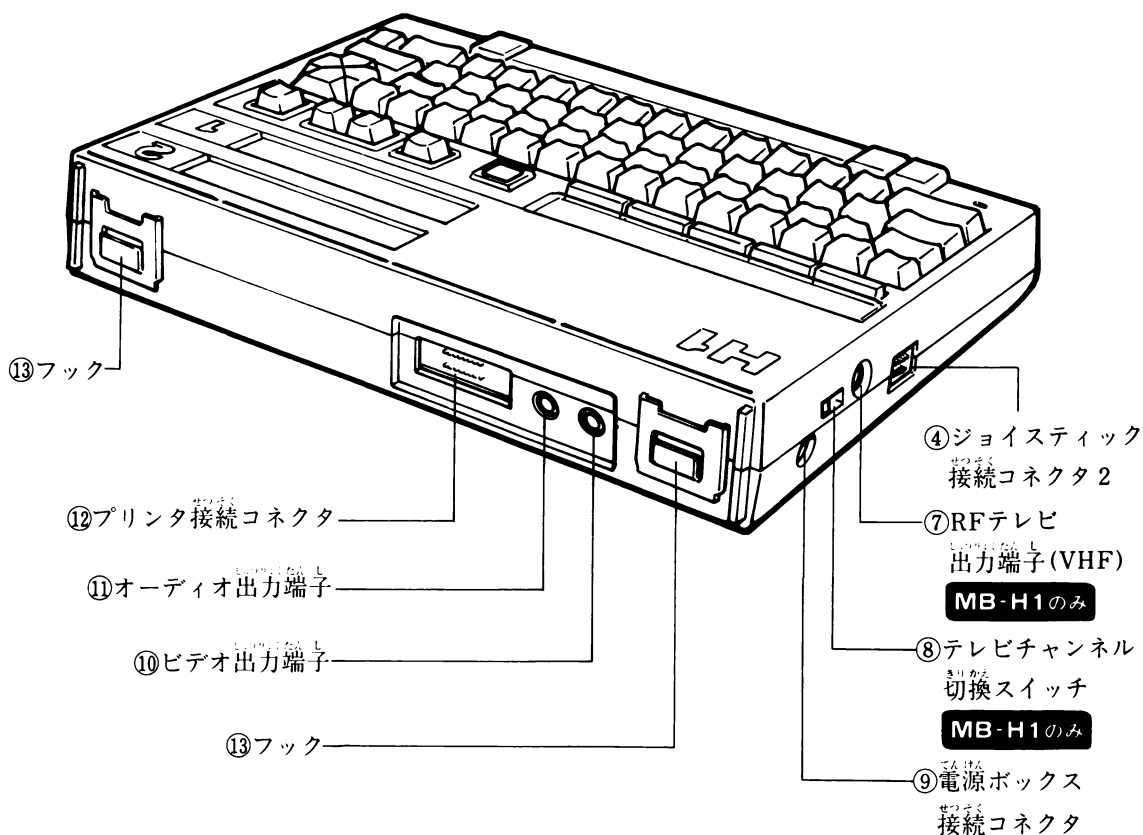
カセットテープレコーダのかわりにコンパクトフロッピーディスクドライブを接続したり、プリンタを付け加えると一層充実したシステムになります。(H1 システムアップ構成をご覧ください。)

III どうすれば動くの (接続と組み立て)

1. 各部の名称とはたらき

外形図





①電源スイッチ

H1の電源を入れたり切ったりします。一度“切”にしてからすぐに“入”にすると、H1が間違った動作をすることがありますので、“切”にした場合には、5秒以上たってから“入”にしてください。

②電源表示ランプ

H1の電源が入っているとランプが点灯し、電源を切ると消えます。

③ハンドル

本体を持ち運ぶときに、手前に引き出して使用します。使わないときは、押し込んでしまっておきます。

④ジョイスティック接続コネクタ

(9ピン)

本体の左右にジョイスティック (別売) を2個まで接続することができます。

⑤カセットレコーダ接続コネクタ

(8ピン)

カセットレコーダを接続します。接続には、付属のカセットレコーダ接続ケーブルを使用してください。

⑥スピードコントロールスイッチ

ゲームのスピードを3段階に変えることができます。ノーマル1、スロー2、スロー3の3段階で、この順にスピードが遅くなります。

ゲームを楽しむとき以外は、ノーマル1にしてください。

⑦RFテレビ出力端子(VHF) MB-H1のみ

ビデオ入力端子のない家庭用テレビと接続するときには、付属のテレビ接続ケーブルとスイッチボックスを使用してください。

H1からはテレビ1チャンネルまたは、2チャンネルに相当する信号が出ます。

⑧テレビチャンネル切換スイッチ MB-H1のみ

あなたの地域の放送局と同じチャンネルを使用すると、画面にし模様が出ますのでチャンネルは、放送局が使用していない方を選んでください。ボールペンなどの先で切り換えてください。(スイッチはあらかじめ「2」に設定されています。)

例) 放送局が1チャンネルを使用している場合

テレビチャンネル切換スイッチ……2

テレビのチャンネル……2 (VHF)

⑨電源ボックス接続コネクタ(7ピン)

付属の電源ボックスを接続します。

⑩ビデオ出力端子

テレビのビデオ入力端子(映像入力)と接続します。接続には、付属のテレビ接続ケーブルを使用してください。⑦RFテレビ出力端子(MB-H1のみ)を使用してテレビと接続した場合には、この端子は接続する必要はありません。

⑪オーディオ出力端子

テレビのビデオ入力端子(音声入力)と接続します。

⑦RFテレビ出力端子(MB-H1のみ)を使用してテレビと接続した場合には、この端子は接続する必要はありません。

⑫プリンタ接続コネクタ(14ピン)

プリンタを接続します。接続には、MSX規格に適合した接続ケーブルを使用してください。本機のプリンタインターフェースは、MSX用プリンタを基本に設計してありますので、これ以外のプリンタを接続した場合の動作は保証できません。

⑬フック

本体側と電源ボックス側にあるそれぞれ2個のフックで一体化できます。

ただし本体背面に何も接続しない場合に限りです。

なお、各接続コネクタの方式は、MSX規格で決められていますので、方式に合った接続ケーブルおよび機器を使用してください。

番号のついていない部分、(たとえばキーボード、ファンクションキーなど)は、操作方法で詳しく説明します。

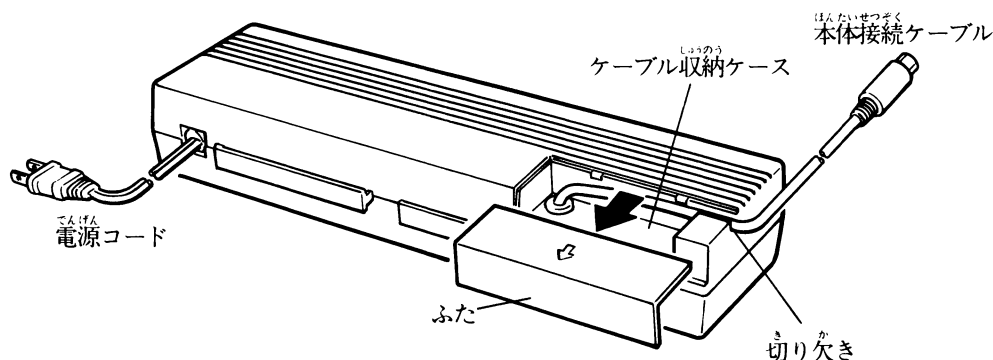
2. 接続のしかた

接続するときには、H 1 やその他の装置の電源は全て、
「切」にしてください。

(1) 電源ボックスとの接続

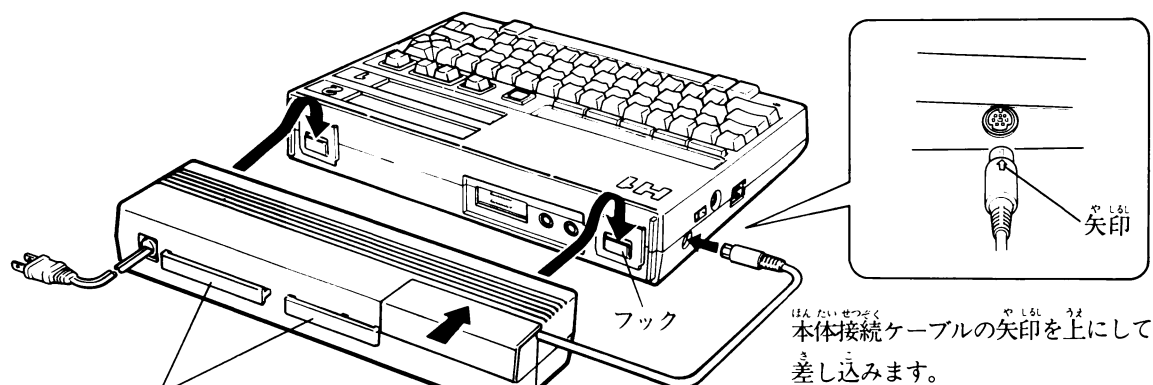
- ① 電源ボックスのケーブル収納ケースのふたを矢印の方向にずらして、とりはずし、中から本体接続ケーブルを引き出します。

電源ボックスは、下図のように安定した置き方で使用しましょう。



- ② 本体接続ケーブルを、電源ボックスのケーブル収納ケースの切り欠きに通し、ふたをもとのようにとりつけた後、H 1 本体の電源ボックス接続コネクタに接続します。

- ③ H 1 を使わないときには、本体接続ケーブルを H 1 からとりはずし、軽く巻いてケーブル収納ケースにしまっておきましょう。



電源コード巻き取り部
電源コードは電源コード巻き取り部に軽く巻くと持ち運びに便利です。

切り欠き

H 1 本体の後ろ側のコネクタ（ビデオ出力端子やプリンタ接続コネクタ、オーディオ出力端子）に何も接続しないときには、フックにより H 1 本体と電源ボックスは、一体化して使用することができます。

(2)テレビとの接続

MB-H1は[A]の方法(本ページ)または[B]の方法(次ページ)いずれかの方法で接続してください。

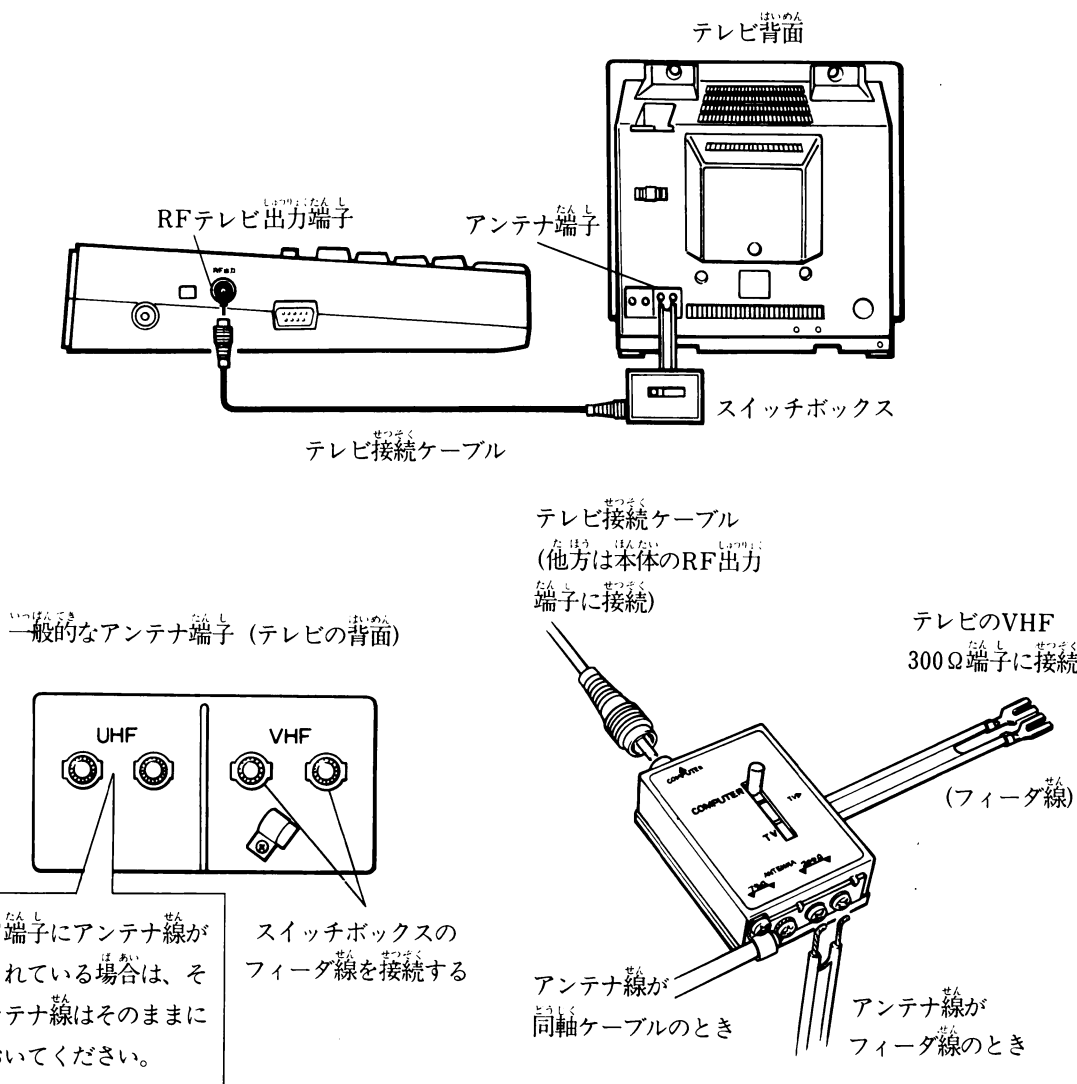
MB-H1Eは、次ページの[B]の方法で接続してください。

[A]ビデオ入力端子のないテレビの場合 MB-H1のみ

まず、付属のスイッチボックスのフィーダ線をテレビの裏にあるアンテナ端子につなぎます。端子はVHF300Ωを使用します。端子がVHF75Ωしかない場合は、市販の300Ω-75Ω変換器を取り付けてつないでください。

VHF端子が75Ω300Ω両用の場合は、必ず300Ωに切り換えてつないでください。

チャンネルの切り換えについては26ページの“テレビチャンネル切換スイッチ”をご覧ください。

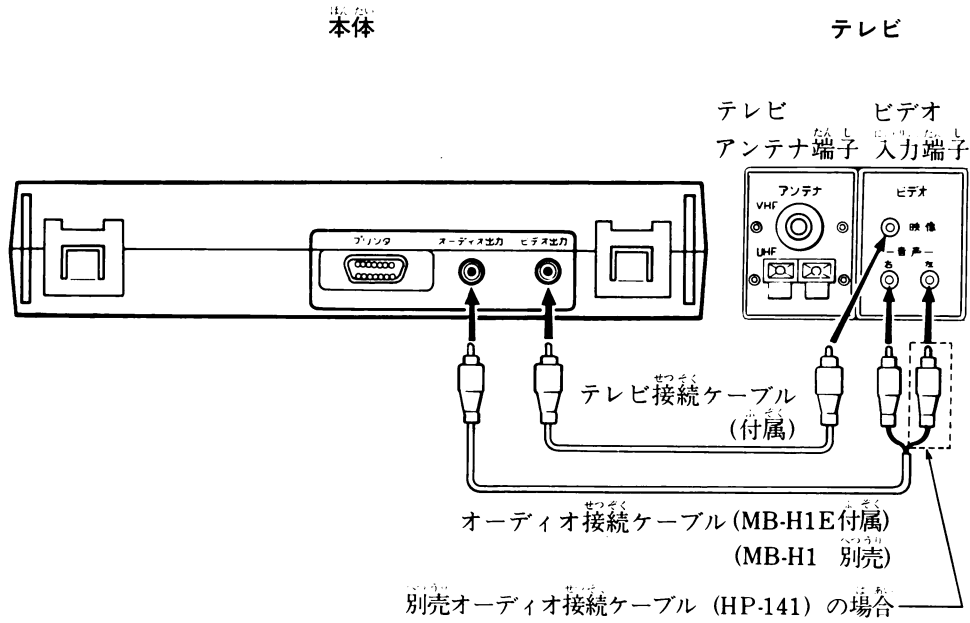


スイッチボックスをTV側に切り換えてテレビ番組も簡単に見ることができます。このとき、H1の影響でテレビに雑音が入る場合がありますのでH1の電源は切ってください。

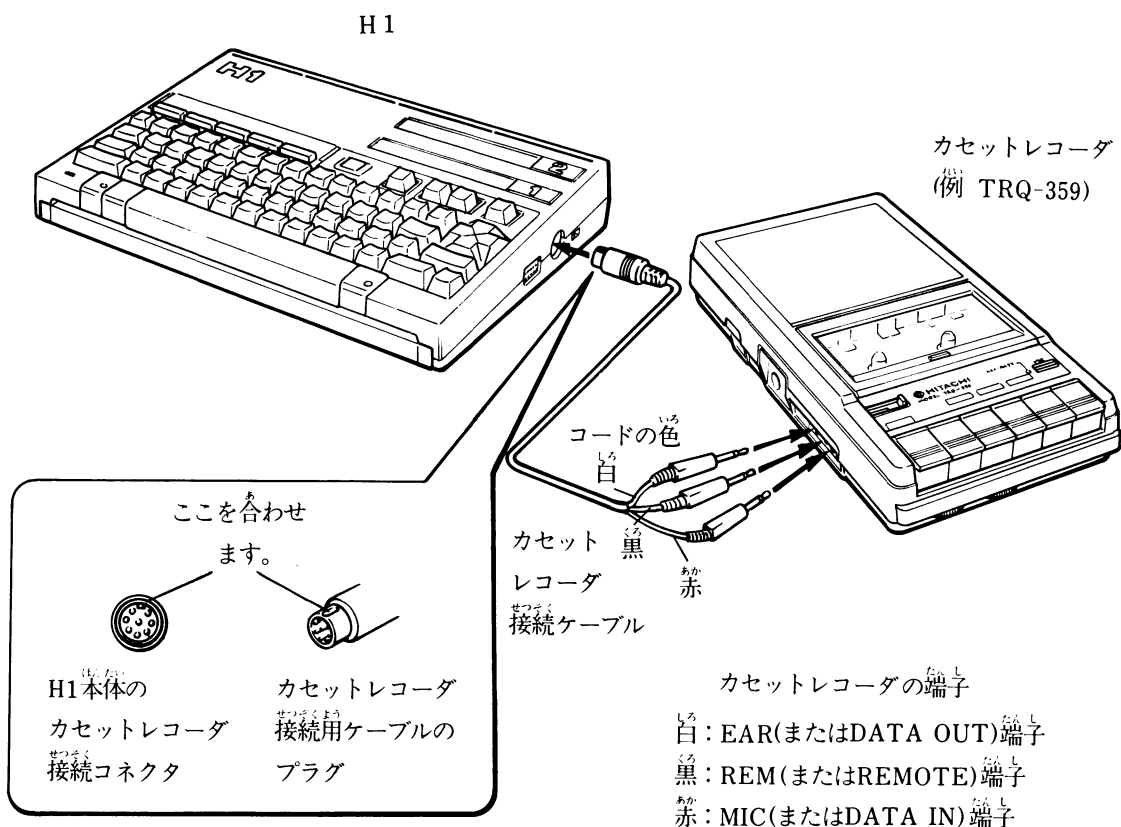
④ビデオ入力端子のついているテレビの場合

ビデオ入力端子付きテレビと本体の接続には付属のテレビ接続ケーブルを使用します。

なお、本体のオーディオ出力端子と、テレビの音声入力端子の接続はオーディオ接続ケーブルで接続します。



(3)カセットレコーダとの接続



①リモートコントロールの使用法

カセットレコーダのREM(リモート)端子に、カセットレコーダ接続ケーブルの黒のコード(リモート端子接続用)を接続すると、H1からカセットレコーダをリモートコントロールすることができます。カセットレコーダを再生または録音状態にして、CLOADやCSAVEというベーシックの命令を実行するとカセットレコーダは自動的にスタートし、実行が終わると自動的に止まります。

ところが、このままでカセットテープの巻き戻しや早送りをすることができません。早送りや巻き戻しを行なうときは、REM端子に接続したプラグをはずして行なうか、ベーシックのMOTOR ON/OFF命令を実行してください。

②リモートコントロールを使用しないとき

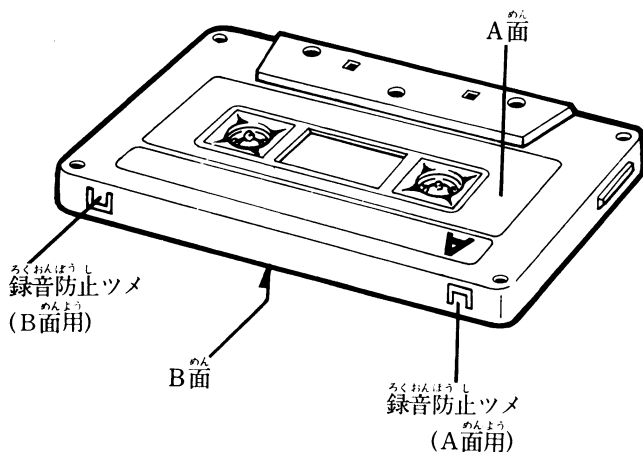
REM(リモート)端子を使用しないときにはカセットレコーダの自動スタート、自動ストップはできません。記録するときは、CSAVEを実行する前に、カセットレコーダを録音状態にし、カセットテープに記録したプログラムを読みこむときには、CLOADとしてから再生状態にしなければなりません。

③ 音量調節とトーンコントロール

TRQ-359 などのカセットレコーダに音量調節およびトーンコントロールのつまみがあります。他のカセットレコーダを使用するときには正確にデータ、プログラムが入出力できるように音量つまみとトーンコントロールつまみを調節してください。TRQ-359 の場合には正確にデータ、プログラムの入出力を行なうために音量調節つまみは5～7、トーンコントロールつまみは7～10の表示のあるところで使用します。

④ プログラムの保護

普通のカセットテープに録音するときと同じように、カセットテープのすでにプログラムが記録されている部分にさらに新しく記録をすると、それ以前に記録されていた内容は消されてしまいます。大切なプログラムを誤って消したりしないように、記録するときは十分に注意してください。カセットテープの録音防止用ツメを折りとっておくと、誤って記録したりすることを防止することができます。



⑤ ボーレートの設定

H1とカセットレコーダでプログラムやデータのやりとりをするときに、1秒間に何個のデータを送り出せるかを表わす単位をボーレートといいます。

H1はベーシックの命令で、2種類のボーレートを切り換えて使用することができます。(ベーシックのCSAVE命令または、SCREEN命令)

●1200ボーモード

電源スイッチを入れると自動的に1200ボーモードに設定されます。

●2400ボーモード

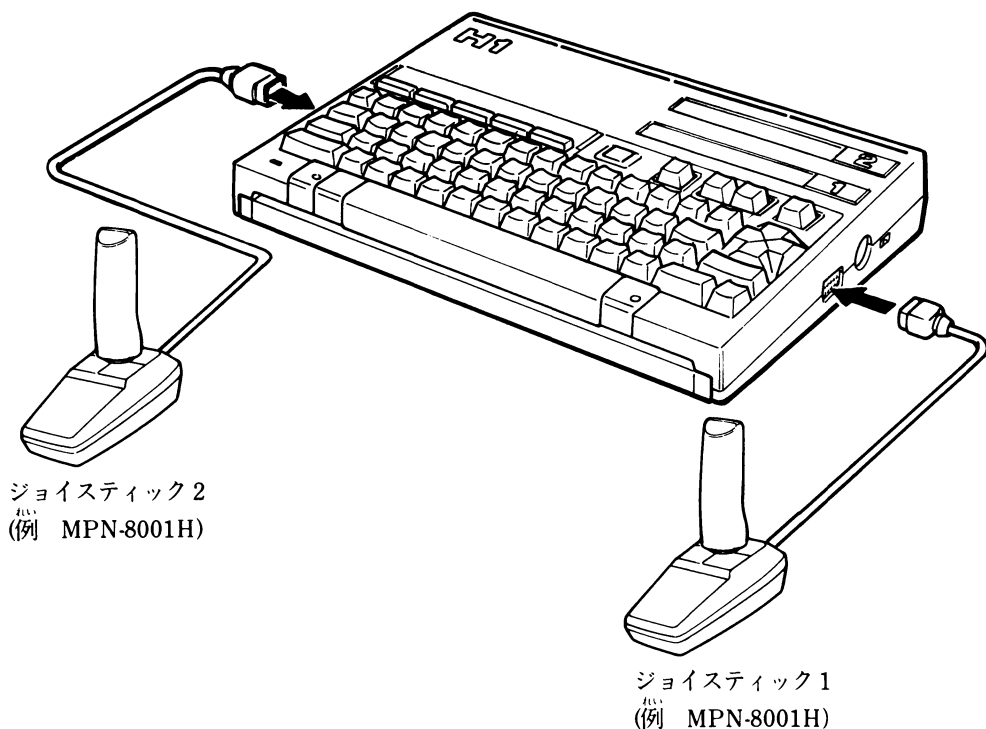
ベーシックのSCREEN命令またはCSAVE命令で設定します。このモードでは1200ボーモードに比べて実行速度は早いのですが、使用するカセットレコーダの種類によっては正常に再生できないことがあります。このようなときには1200ボーモードで使用してください。

●カセットレコーダからデータを読み込むときには、H1が自動的にボーモードを判断しますので、切り換えの必要はありません。

●記録するときと、再生するときには、できるだけ同じカセットレコーダを使用してください。

(4) ジョイスティック^{へつうり}（別売）との接続^{せつぞく}

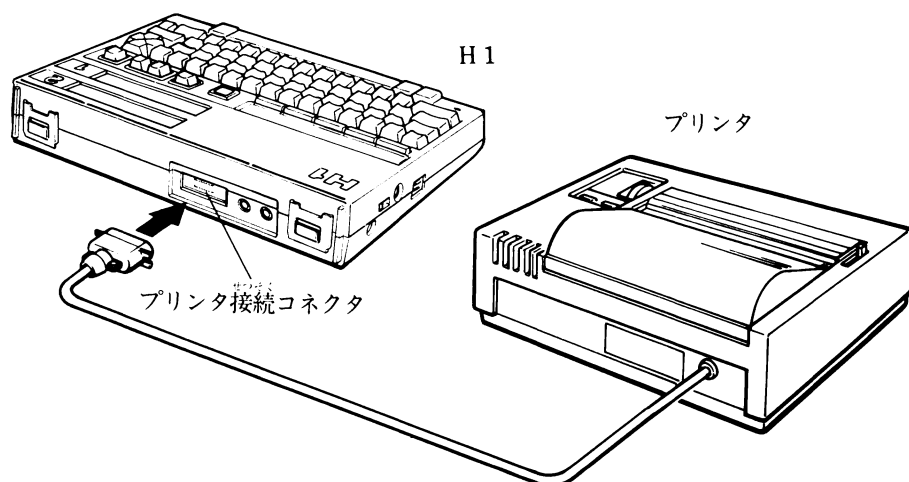
ジョイスティックのケーブルをH1の^{カニゴクめん}両側面のジョイスティック^{せつぞく}接続コネクタに接続します。
接続のジョイスティックは、MSX規格に^{きかく}適合したものを^ご使用ください。



ジョイスティックは、H1のジョイスティック^{せつぞく}接続コネクタ1、2のどちらに接続してもかまいませんが、カートリッジやソフトテープの^{せつぞく}説明書で指定されている場合には、そのコネクタに接続します。

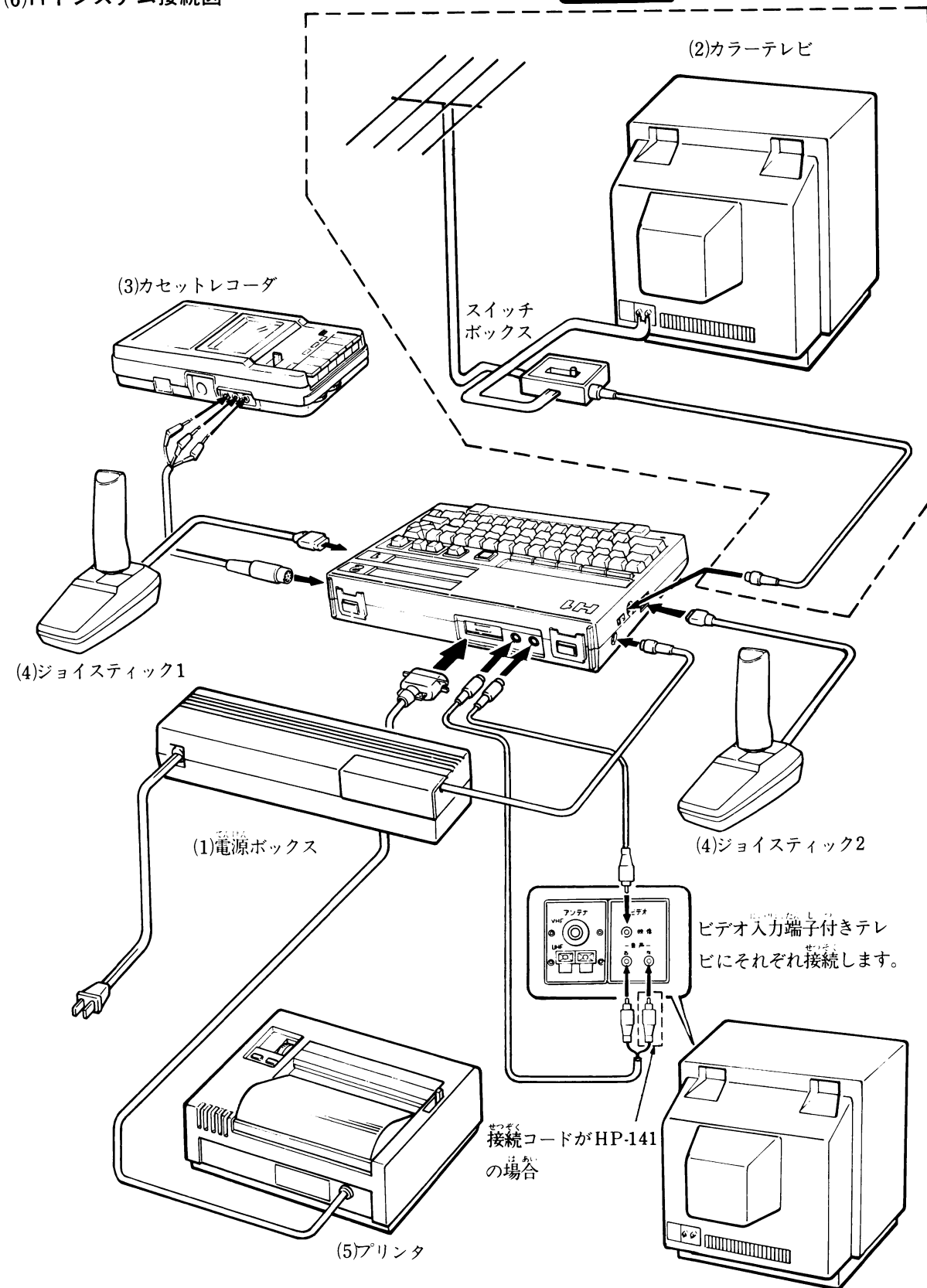
(5) プリンタの接続^{せつぞく}

MSX規格に^{きかく}適合したプリンタおよびプリンタ^{せつぞく}接続ケーブルを^ご使用ください。



(6) H 1 システム接続図

MB-H1のみ



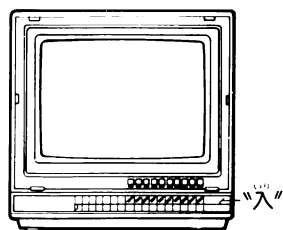
IV さあ動かしてみよう

1. 電源の入れ方

カートリッジが取り付けられているときは、
まずそれを抜きます。

正しく接続されているかどうか27ページ
から33ページをご覧ください。

テレビなどの周辺
機器の電源を入
れます。



テレビのビデオ
入力に接続した
場合

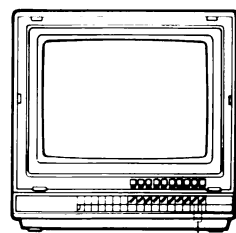
スピードコントロールスイッチを“ノーマル1”の位置

ビデオ入力のないテレビ
の場合(アンテナ端子に
接続した場合)

MB-H1のみ

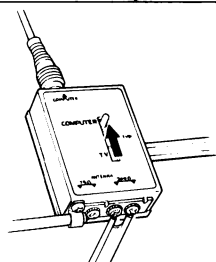
入力切換スイッチを“ビデオ”にします。

テレビのチャン
ネルを26ページで説
明しましたH1の
テレビチャンネル
切換スイッチ(1ま
たは2)に合わせます。



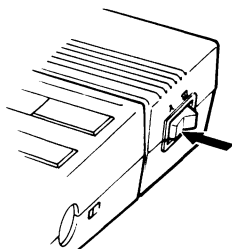
1チャンネルか
2チャンネル

テレビのアンテナ
端子に接続したス
イッチボックスを
COMPUTER 側
にします。



スイッチボックス

H1の電源を入
れます。



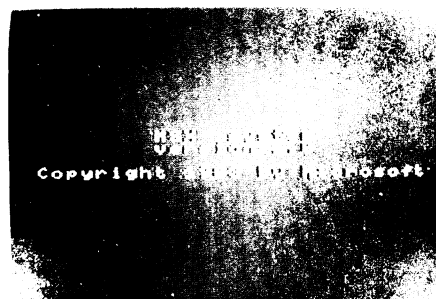
使い終わったらH1の電源を切ります。

テレビなどの周辺機器の電源を切ります。

2. H1が動きはじめる^{うご}

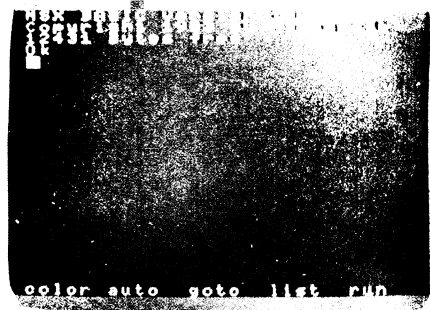
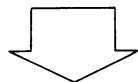
MB-H1Eの場合^{は あい}

電源^{でんげん}“入^{いり}”にすると、H1が動きはじめます。H1を動かす場合^{うご かい}、テレビの取扱説明書^{とりあつかいしゅ}にしたがって、テレビ画面^{てれび かいめん}の調整^{ていせい}を行ない、もっとも見やすいように合わせてください。また終わって、テレビ放送^{てれび ほうそう}を見る場合^{み る かい}、多少、調整位置^{ていせい いち}が異なっていると思われますので、再び調整^{ていせい}を行なってください。



電源^{でんげん}を“入^{いり}”にすると、一番最初^{いちばん さいしょ}に約3秒間表示^{ひょうし}されます。

画面 1



これで、いつでもベーシックを始められる状態^{じたい}になります。

画面 2

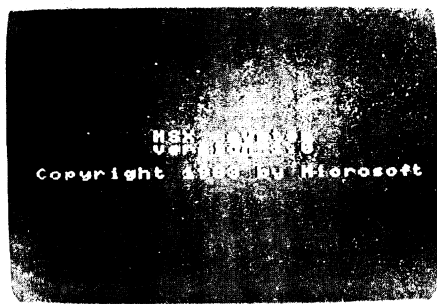
39ページからの「キーボードの扱い方^{あつかい ほう}」をご覧^みになってキーボードの練習^{れんしゅう}をしてみてください。

カートリッジを使用^{もち}する場合は、一度電源^{でんげん}を切^きってから、38ページの「カートリッジの扱い方^{あつかい ほう}」を参照^{さんしょう}してください。

●デモプログラム MB-H1のみ

電源^{でんげん}“入^{いり}”^{いり}にすると、MB-H1の大きな特長^{とくちよう}のひとつ「内蔵^{ないざう}プログラム」が動きはじめます。H1を動か^{うご}かす場合^{ばあひ}、テレビの取扱^{とりあつか}説明書^{せうめいしょ}にしたがって、テレビ画面^{かめん}の調整^{ちようせい}を行^おない、もっとも見やすいように含^ふわせてください。また終わ^おって、テレビ放送^{ほうそう}を見る場合^{ばあひ}、多少^{たうしょう}、調整位置^{ちようせい いち}が異^{こと}なっていると思^{おも}われますので、再^{ふた}び調整^{ちようせい}を行^おなってください。

では、しばらくの間^{かん}、キーボードに触^ふれるのは待^{まち}って画面^{かめん}を見てみましょう。



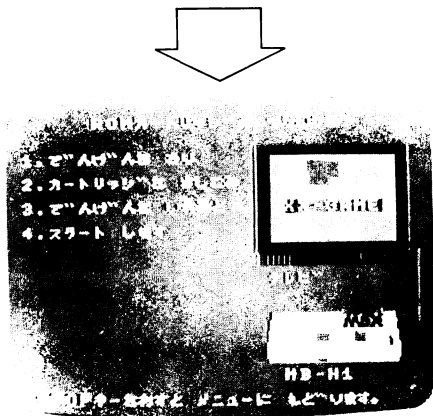
画面 1

電源^{でんげん}を“入^{いり}”にすると、一番最初^{いちばんさいしょ}に約3秒間^{やく 3びょうかん}表示^{ひょうじ}されます。



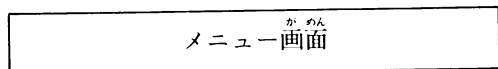
画面 2

メニュー画面^{かめん}といひます。
約40秒間^{やく 40びょうかん}待^{まち}ってみましょう。
画面^{かめん}の下^{した}のROM^{てんめつ}が点滅^{てんめつ}します。

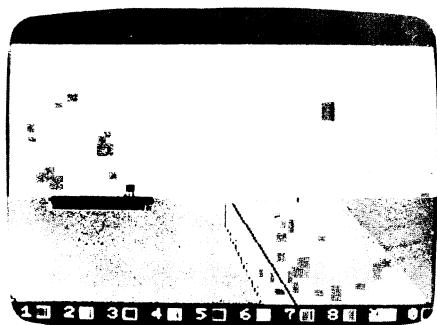


画面 3

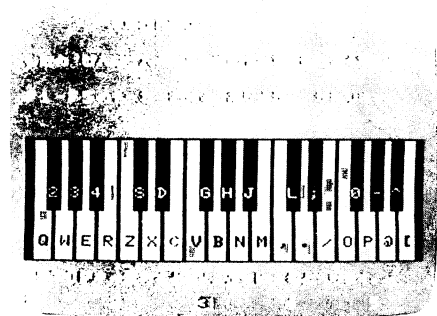
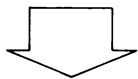
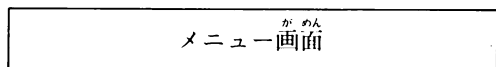
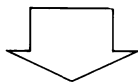
ROMカートリッジ^{つか かた づはじ}の使^{つか}い方^{かた}が表示^{ひょうじ}されます。



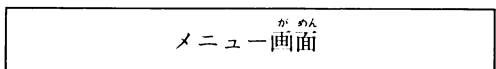
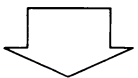
メニュー画面^{かめん}に戻^{もど}ります。
スケッチ^{てんめつ}が点滅^{てんめつ}します。



がめん
画面 4



がめん
画面 5



スケッチプログラムが実行されます。
そして、「海の風景」を描き始めます。
描き終わってから約20秒間表示しています。
注) 絵を描いている最中にキーを押すと、
正しく動作しない場合があります。

メニュー画面に戻ります。
サウンドが点滅します。

サウンドプレイをするための画面が表示されます。
そして「アルルの女」を自動演奏します。

メニュー画面に戻ります。
ROMが点滅します。

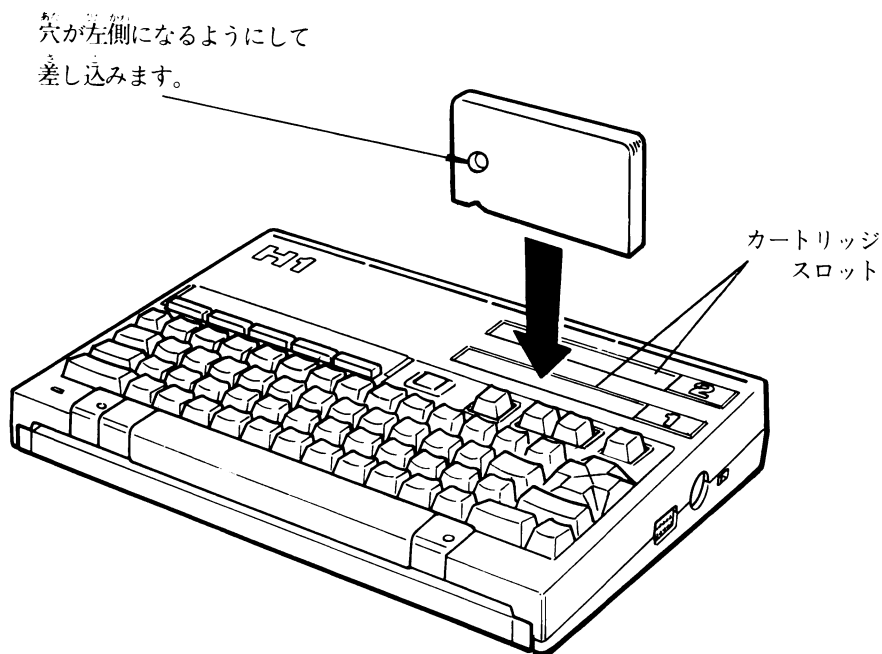
★何もキーを押さなければ、このように画面2～5をくり返します。

途中でキーを押すと、その状態で止まる場合がありますが、もう一度、くり返して実行させたい場合は、電源を一度“切”にした後、約5秒後に再び“入”にしてください。なおくり返しの画面の途中からメニュー画面にもどる場合は **STOP** キーを押してください。

メニュー画面で止まったままになります。

各プログラムを実行する方法は、54ページの「MB-H1のメニュー画面の使い方」を参照してください。

3. カートリッジの扱い方



H1には、2つのカートリッジスロットがあります。通常はスロット1にゲーム用のカートリッジなどを差し込みます。カートリッジを使わないときにはふたがされていますが、カートリッジの正面（左に穴が見える面）を手前向きにして、ふたを押し開きながらまっすぐ差し込んでください。

カートリッジの抜き差しは必ず電源を切った後、行なってください。

またふたを開くとリセットされるようになっていますので、プログラムを作成中、誤ってふたを開くと作ったプログラムが消えてしまいますので、注意してください。

H1には、2個のカートリッジを差し込むことができますが、通常は、スロット1の方が優先されます。

スピードコントロールスイッチは、ゲームなどが実際に始まってから、スロー2、スロー3に切り換えてください。またゲームによってはスピードコントロールスイッチのスロー2、スロー3が適さないものもありますのでご注意ください。

ゲーム以外の動作時には、必ずノーマル1でご使用ください。

カートリッジは、**MSX** マークのついたものを使用してください。

4. キーボードの扱い方

ここではキーボードの扱い方を説明します。電源を入れてメニュー画面が出たらファンクションキー **[F1]** を押してベーシックモードにしましょう。(MB-H1Eは、直接ベーシックモードになります。)H1に何かをさせようとするときには、キーボードから命令を伝えてやらなければなりません。キーボードは、私たちと、H1を結びつけるとても大切な役割をしてくれます。

H1に、キーボードのキーを押して命令を伝えることを「入力する」といいます。なお、キーは3つ以上を同時に押しますと、正しいキー入力は保証されません。H1のキーボードには、2つの便利な機能がついています。

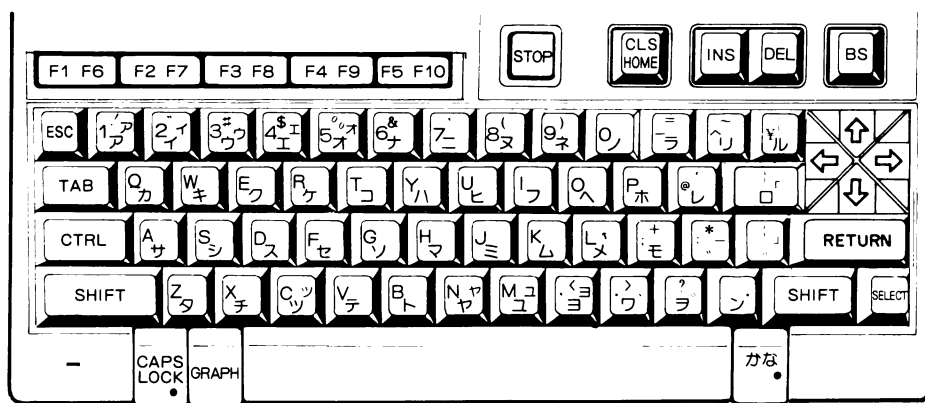
●リピート機能

キーを押し続けると、連続して入力することができます。キーボードの左下にある **[Z]** キーを押してみましょう。軽く1回押すと画面には、「Z」が表示されますね。しばらくの間、押しつけてみましょう。「ZZZ………」といくつも続けて表示されますね。このような機能をリピート機能といいます。同じ文字を続けて入力したり、カーソルをはなれたところに移動させるときにとっても便利です。

●先行キー入力機能

プログラム実行中にキーボードから入力する文字が画面に表示されない状態でも、キー入力を事前に受け付けることができます。先行キー入力は39字まで、受け付けます。ただし、**[CTRL] + [STOP]** をキー入力したときには、それまでに先行入力した内容は消えてしまいます。なお、周辺機器の動作中に入力しても、先行入力できない場合があります。

1 キーのならばかた



H1のキーは、次のようにわけることができます。

1. 文字キー
 - 英字キー
 - かな文字キー
 - 数字キー
 - 記号キー
 - グラフィック文字キー
2. 特殊キー
 - モード切りかえ用のキー
 - コントロール用のキー
3. ファンクションキー

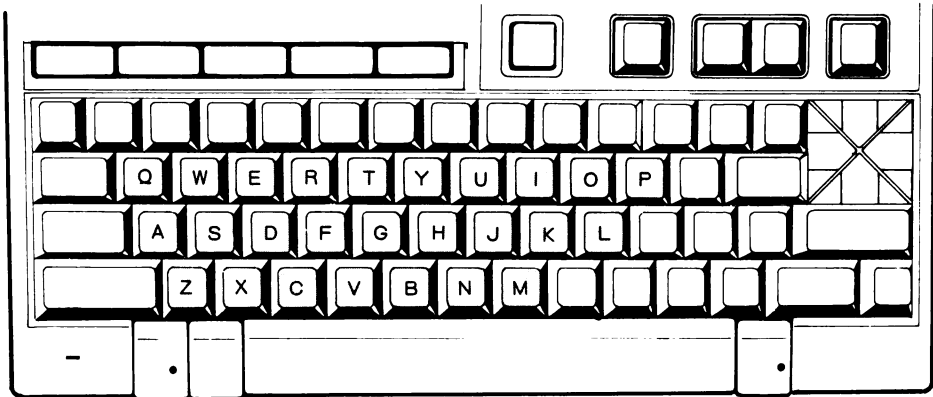
これから、それぞれのキーの扱い方を説明します。

② 文字キー

H 1 に命令を伝えるのに、最もよく使います。入力される文字の種類によって、英字キー、かなキー、数字キー、記号キー、グラフィック文字キーに分けることができます。

① 英字キー

図中、色の濃い部分が英字キーです。

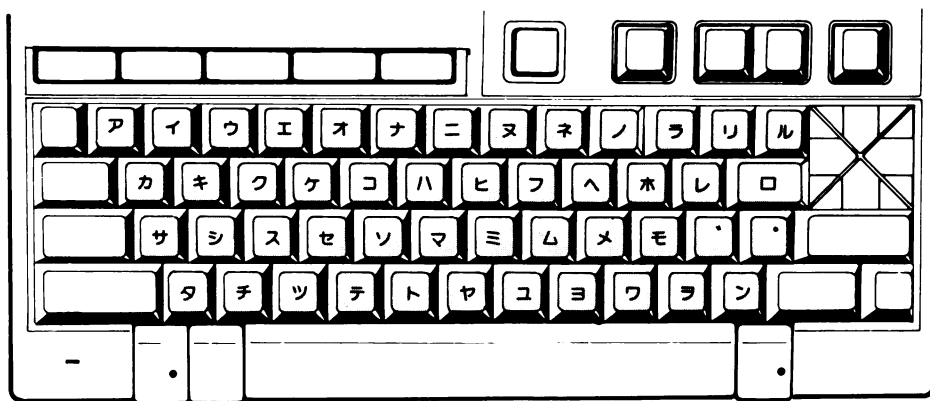


○ **かな** キーのランプと **CAPS LOCK** キーのランプが消えている状態で、英字キーを押すと、英字の小文字が入力できます。
SHIFT キーを押しながら、英字キーを押すと、英字の大文字を入力することができます。**CAPS LOCK** キーを押して、ランプをつけた状態で英字キーを押すと、英字の大文字を入力でき、このときに **SHIFT** キーを押しながら英字キーを押すと、英字の小文字が入力できます。もう一度、**CAPS LOCK** キーを押すと、ランプが消え、もとの状態に戻ります。

② かな文字キー

かなキーを押して、かなキーのランプをつけます。

図中、濃い部分がかな文字です。



● ひらがな

CAPS LOCK キーのランプが消えた状態でかな文字キーを押すと、ひらがなを入力することができます。SHIFT キーを押しながらかな文字キーを押すと、ひらがなの小文字（あ、い、う、え、お、つ、や、ゆ、よ）を入力することができます。この他のひらがなの小文字はありません。

<例>

1!ア → あ (大文字)

SHIFT + Aサ → さ (大文字)
(小文字はないからです。)

SHIFT + 1!ア → あ (小文字)

● カタカナ

CAPS LOCK キーのランプとかなキーのランプがついている状態でかな文字キーを押すと、カタカナを入力することができます。SHIFT キーを押しながら、かな文字キーを押すと、カタカナの小文字（ア、イ、ウ、エ、オ、ツ、ヤ、ユ、ヨ）を入力することができます。この他のカタカナの小文字はありません。

<例>

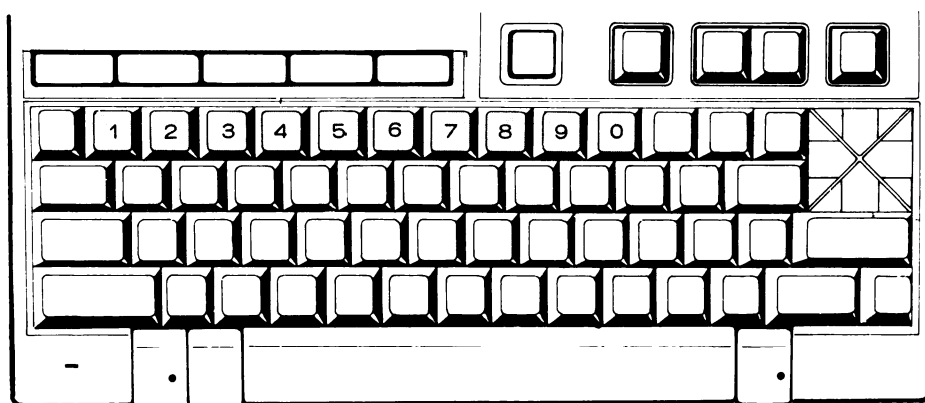
1!ア → ア (大文字)

SHIFT + Aサ → サ (大文字)
(小文字はないからです。)

SHIFT + 1!ア → ア (小文字)

③ 数字キー

図中、濃い部分が数字キーです。

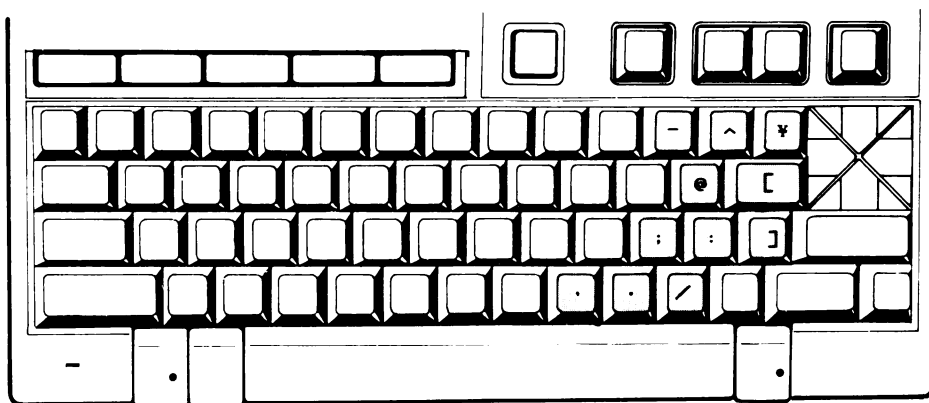


かな キーのランプが消えている状態で、数字キーを押すと、数字を入力することができます。

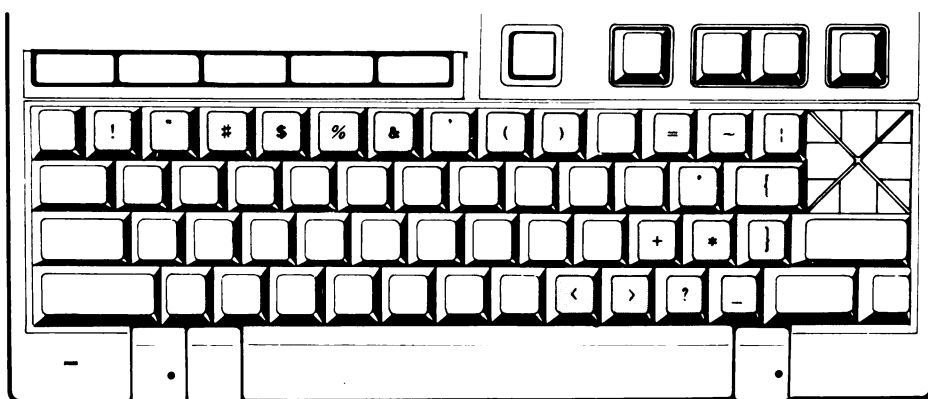
④ 記号キー

図中、濃い部分が記号キーです。

● 英記号



かなキーのランプ、CAPS LOCK キーのランプが両方とも消えている状態で、英記号キーを押すと、上の図の英記号を入力することができます。SHIFT キーを押しながら、英記号キーを押すと、下の図の英記号を入力することができます。

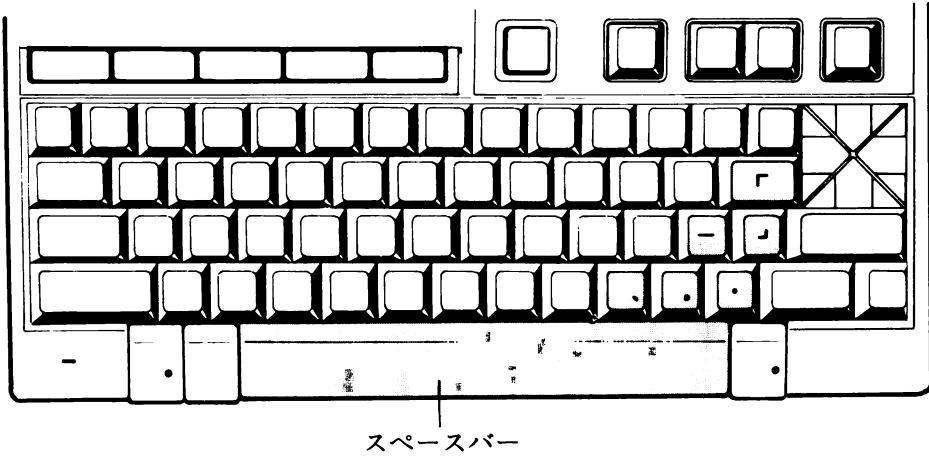


<例>

¥ | ル → ¥


SHIFT + ¥ | ル → |

●その他の記号キー



かなキーのランプがついている状態で、**SHIFT** キーを押しながら、記号キーを押すと、上の図のような、かな記号を入力することができます。

〈例〉

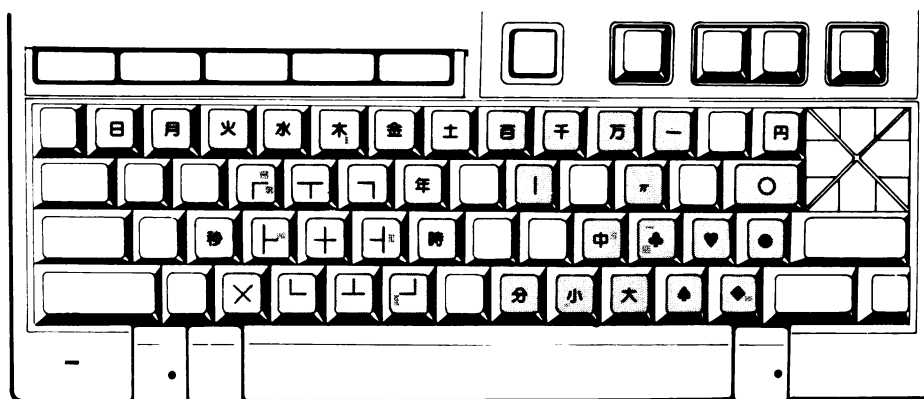
SHIFT +  → 」（かぎかっこです）

 (スペースバー)

このキーを押すと空白文字（スペース）を1文字入力します。


⑤ グラフィック文字キー


図中、濃い部分がグラフィック文字キーです。



グラフィックキーを押しながら、グラフィック文字キーを押すと、上の図のような漢字やグラフィック文字を入力することができます。

〈例〉

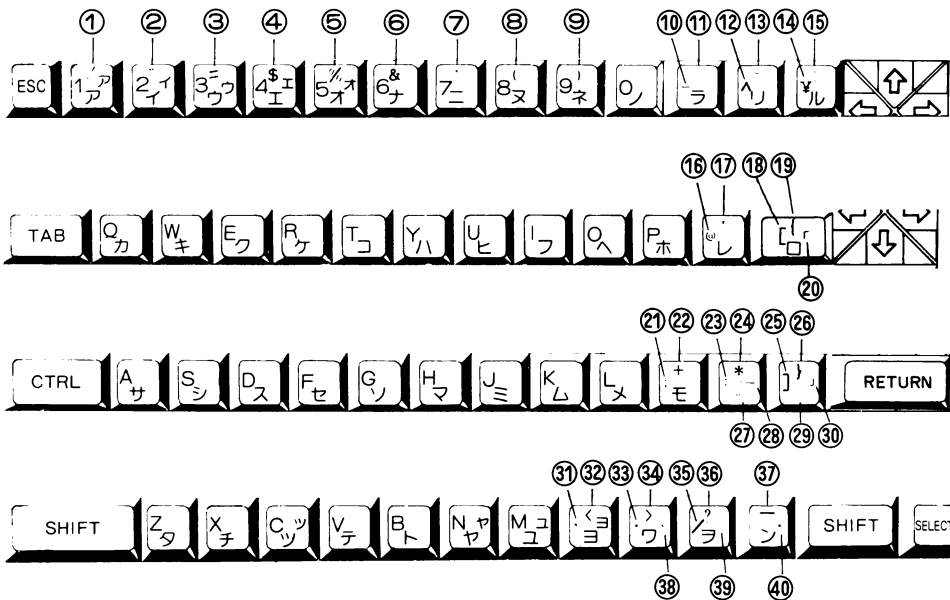
GRAPH +  → 日

GRAPH +  → ●

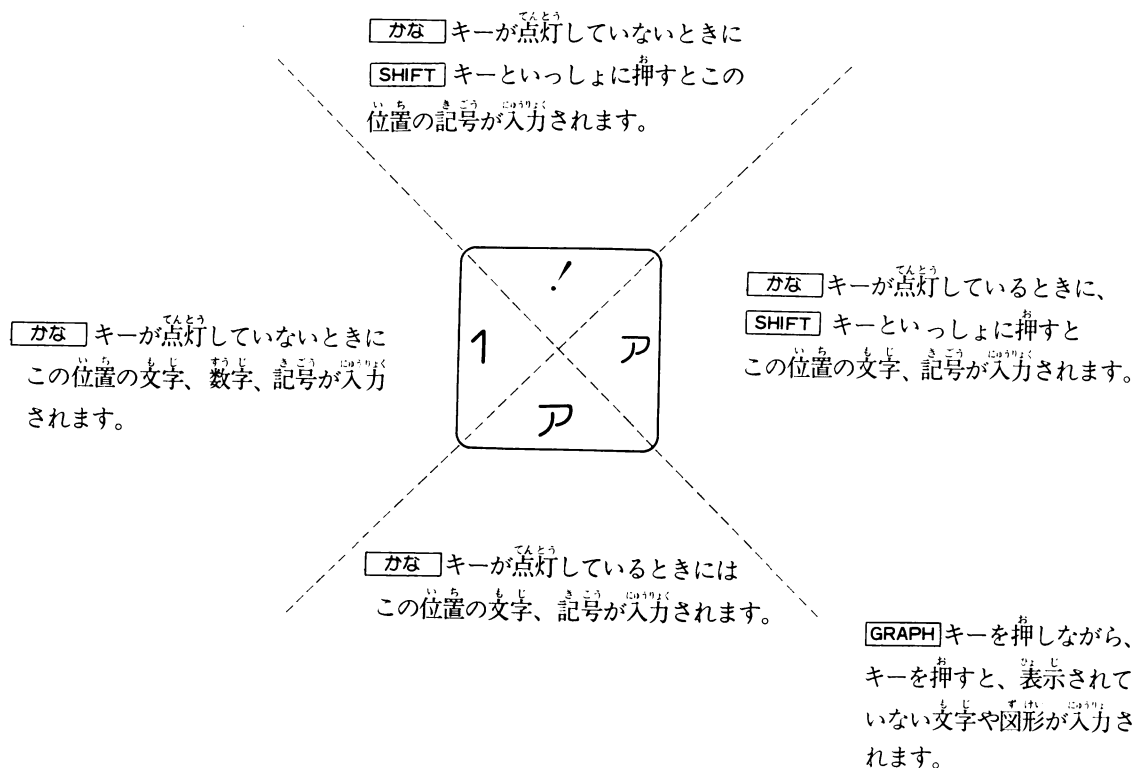
付属のグラフィック文字シールをはがして、上の図を見ながらキーの前面に貼ってください。

⑥文字キーのまとめ

●特殊記号の名称（読み方）



No.	記号	名称（読み方）	No.	記号	名称（読み方）
①	!	感嘆符、イクスクラメーション	②①	;	セミコロン
②	"	ダブルクォート	②②	+	プラス
③	#	シャープ、ナンバー	②③	:	コロ
④	\$	ドル記号、ダラー	②④	*	アスタリスク、星印
⑤	%	パーセント	②⑤]	右大かっこ、ライトブラケット
⑥	&	アンパーサンド、アンド	②⑥		右中かっこ
⑦	'	アポストロフィ、シングルクォートライト	②⑦	"	濁点
⑧	(左かっこ、レフトバレンス	②⑧	—	長音記号
⑨)	右かっこ、ライトバレンス	②⑨	°	半濁点
⑩	-	マイナス、ハイフン（②⑩より短い）	③⑩	」	右カギかっこ
⑪	=	等号、イコール	③①	,	カンマ
⑫	^	指数記号、アップアロー	③②	<	不等号（より小）、レスザン
⑬	~	フィールドマーク、ウェーブ	③③	.	ピリオド
⑭	¥	円記号	③④	>	不等号（より大）、グレーターザン
⑮		分離記号	③⑤	/	スラント、スラッシュ
⑯	@	アットマーク、単価	③⑥	?	疑問符、クエスチョン
⑰	`	シングルクォートレフト	③⑦	—	アンダーライン
⑱	[左大かっこ、レフトブラケット	③⑧	,	読点
⑲		左中かっこ	③⑨	。	句点
⑳	「	左カギかっこ	④⑩	・	中黒、中点、センターピリオド



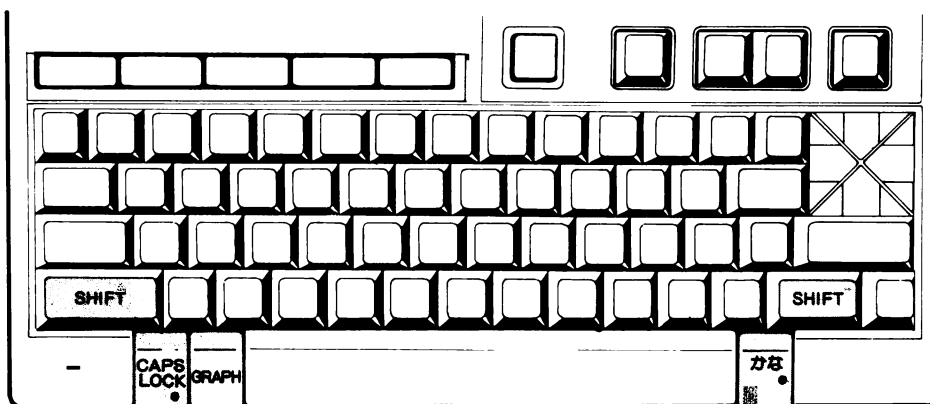
●間違った使いやすいキー

- | | | |
|--|--|--|
| | | アポストロフィ REM命令の省略形として使います。 |
| | | カンマ ベーシックの中では区切りとしてよく使います。 |
| | | マイナス記号 引き算をするときやベーシック命令中で使います。
(長音記号より短い) |
| | | 長音記号 かな記号です。「ベーシック」というように音をのばすときに使います。 |
| | | アンダーライン 下線を引きます。 |
| | | ピリオド 英語の文章の終わりや、小数点に使います。 |
| | | 中黒 言葉の区切りなどに使います。 |
| | | 半濁点 パ、ピなど半濁音のときに使います。 |
| | | 句点 日本文の終わりに使います。 |

③ 特殊キー

① モード切り換え用のキー

キーを押したとき、かなを入力できる状態を、かなモードといいます。英数文字の入力ができる状態を、英数字モードといいます。このようなモードを切り換えるために使用するのが、つぎの図のキーです。



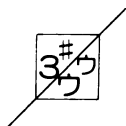
SHIFT キー (シフトキー)

キーボードの両側にあります。どちらのキーを押しても働きは同じですから、どちらか一方を押しながら他のキーといっしょに使用します。

英字モードでは、英文字と記号を入力することができます。かなモードでは、ひらがな、カタカナの文字とかな記号を入力することができます。

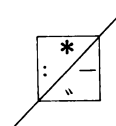
〈例〉

英数字モード



かなモード

英数字モード



かなモード

かな キー（かなキー）

英数字モードとかなモードを切り換えます。押すと、ランプがついてかなモードになり、もう一度押すと、ランプが消えて、英数字モードになります。

GRAPH キー（グラフィックキー）

このキーを押しながら、グラフィック文字キーを押すと、グラフィック文字を入力することができます(⇒45ページ)。

SHIFT キーと同じように、このキーだけを押ししても、何も入力することはできません。

CAPS LOCK キー（キャピタルロックキー）

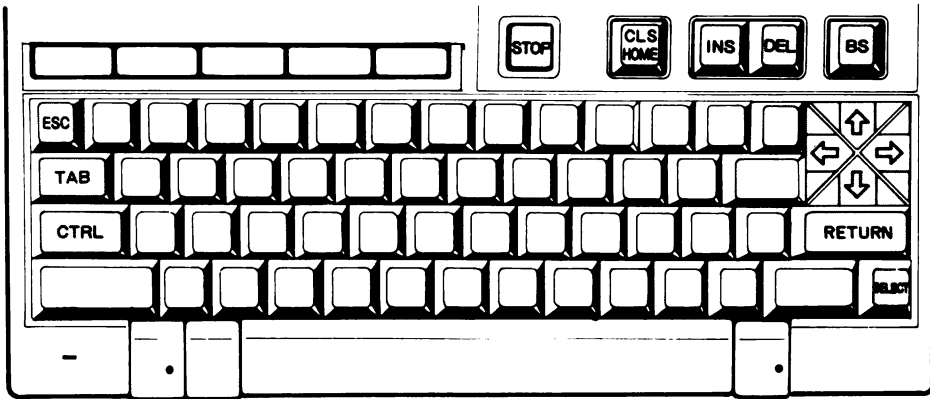
このキーを押すと、ランプがつき、英数字モードでは **SHIFT** キーが押されたままの状態になり、**SHIFT** キーを押さなくても、続けて英字の大文字を入力することができます。（このとき、**SHIFT** キーを押しながら英字キーを押すと、英小文字が入力できます。）かなモードでは、カタカナを入力することができます。

もう一度押すと、ランプが消えて、英小文字（英数字モードの場合）、ひらがな（かなモードの場合）が入力できるようになります。

② コントロール用のキー

コントロールするはたらきをもつキーです。

図中、濃い部分がコントロール用のキーです。



ESC キー（エスケープキー）

ベーシックでは使用しません。

TAB キー（タブキー）

このキーを押すと、カーソルが、左端から9文字目、つづけて押すと、17文字目、25文字目、次の行の1文字目というように、移動します(29文字表示のとき)。また移動した間の文字は消えます。

CTRL キー（コントロールキー）

このキーを押しながら、文字キーなどを押して、H 1に特殊なはたらきをさせます。コントロールキーと組み合わせるキーとそのはたらきは、付録のコントロールコード一覧表(311ページ)をご覧ください。

RETURN キー（リターンキー）

このキーを押すことで、画面に書いたプログラムやデータが、H 1に入力されます。キーボードから命令を打ちこんでも、このキーが押されないと、H 1は何もしません。

このキーが押されると、カーソルは次の行の先頭に移ります。

キー（カーソルコントロールキー）

お押されたキーの矢印の方向にカーソルが移動します。

SELECT キー（セレクトキー）

ベーシックは使用しません。

BS キー（バックスペースキー）

このキーを押すと、カーソルの左側の1文字が消え、その行のカーソルから右側の部分がカーソルとともに左に動きます。

DEL キー（デリートキー）

このキーを押すと、カーソルの文字が消え、その行のカーソルから右側の部分が左に詰まります。

INS キー（インサートキー）

このキーを押すと、カーソルが下半分だけの表示となり、カーソルの左側に文字を割り込ませることができます。もう一度押すと、もとのカーソル表示となり、このモードから抜け出します。（カーソルコントロールキーやリターン キーを押したときも、もとのカーソル表示に戻ります。）

CLS HOME キー（クリアスクリーン・ホームキー）

このキーを押すと、カーソルは左上すみに移動します。**SHIFT** キーといっしょに押すと画面に表示されている文字を消し、カーソルを左上すみに移動させます。

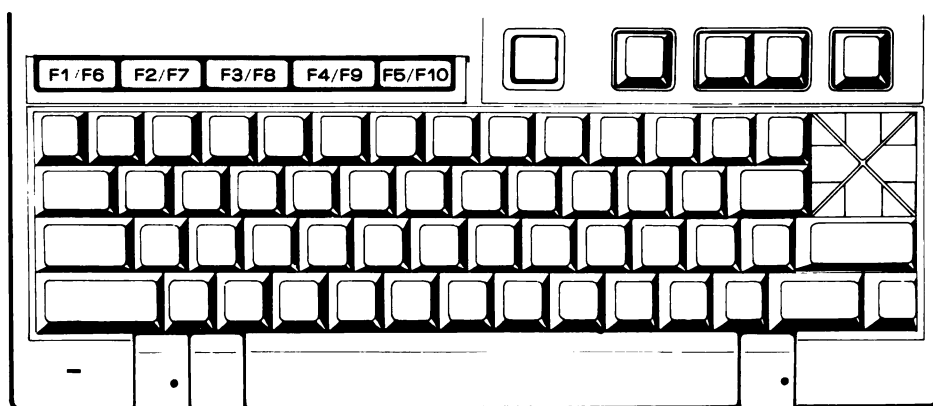
STOP キー（ストップキー）

プログラムの実行を一時停止させます。もう一度押すと一時停止が解除され、プログラムの実行が再開されます。

CTRL キーを押しながらこのキーを押すと、プログラムの実行を中断し、コマンド待ちの状態にします。

④ ファンクションキー

図中、濃い部分がファンクションキーです。



1文字ずつキーを押して入力しなくても、このキーを1回押せば命令を入力することができます。登録されている命令は次のページのとおりです。**[F6] ~ [F10]** は、**[SHIFT]** キーを押しながら **[F1/F6] ~ [F5/F10]** キーを押します。

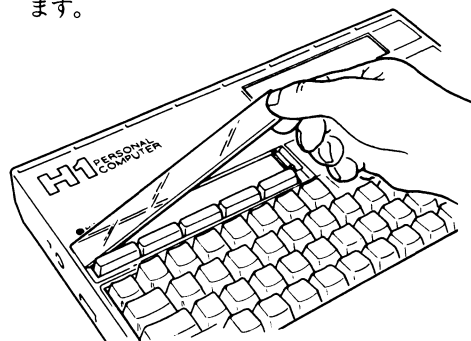
ファンクションキー表示シートの使い方

ファンクションキー表示シートをH1本体に取り付けておくと、ファンクションキー入力の際にとっても便利です。

1. ファンクションキーのすぐ上にある表示シートカバーの右端を硬貨などで軽くこじ開けます。



2. 表示シートカバーを下図のように開け、ファンクションキー表示シートをH1本体に取り付けます。



3. 表示シートカバーをもとのように閉めます。

	命 令	は た ら き
F1	color	画面に色をつける命令です。
F2	auto	行番号を自動発生します。
F3	goto	プログラム実行の流れを変える命令です。
F4	list	プログラムを画面に表示します。(RETURN)キーは登録されていません。)
F5	run (RETURN)	プログラムを実行させる命令です。 ファンクションキーには、(RETURN) キーもいっしょに登録されていますので、ファンクションキーを押すだけで、自動的にプログラムが実行されます。
F6	color 15, 4, 7 (RETURN)	表示されている文字の色を白、背景の色を暗い青、境界の色を水色にします。RUNと同じように(RETURN) キーも、いっしょに登録されています。
F7	cloud"	カセットテープからプログラムを呼び出します。
F8	cont (RETURN)	(CTRL) + (STOP) キーで実行を止めたプログラムを再び実行させます。(RETURN)キーもいっしょに登録されています。
F9	list. (RETURN)	実行が停止している行のプログラムを画面に表示します。エラーが起こったときなどに便利です。(RETURN) キーもいっしょに登録されています。
F10	CLS run (RETURN)	画面を一度きれいにしてから、プログラムを実行します。 (RETURN) キーもいっしょに登録されています。 なおCLSのコントロールコードが入力されているため表示の先頭が1字分ずれています。

なお F1 ~ F5 に登録されている内容は、画面の1番下の行に表示されています。F6 ~ F10 キーの内容は (SHIFT) キーを押すと表示されます。命令の使い方は、ベーシック基礎編をお読みください。

V MB-H1のメニュー画面^{が めん} の使い方^{かた}

MB-H1のみ

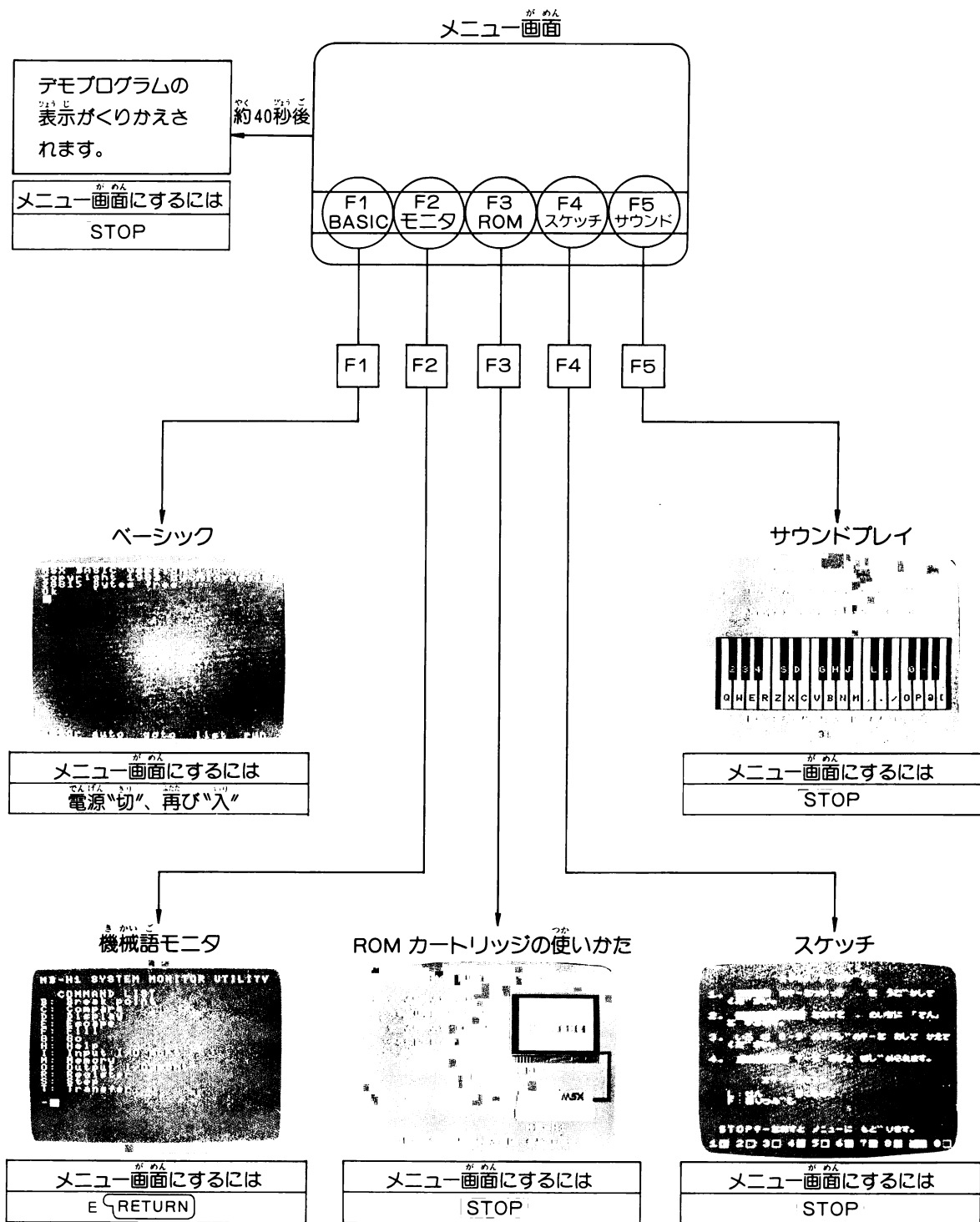
ベーシックやスケッチプログラム、サウンドプレイなどのプログラムをスタートさせるための画面^{が めん}が、メニュー画面^{が めん}です。「H1 が動きはじめる」のところで、たびたび出てきた画面^{が めん}です。



メニュー画面^{が めん}

ファンクションキー（**[F1]** ～ **[F5]**）を押して、画面の下の方に表示されている中の好きなプログラムをスタートさせます。（デモプログラム中のメニュー画面で文字が点滅中にファンクションキーを押すと、色が正しく出ないことがあります。）

約40秒間ファンクションキーを押さなければ、自動的にデモプログラムがスタートします。



※電源^{でんげん}を入れてから最初^{さいしう}に出たメニュー画面^{がめん}は、約40秒^{やく じゅうご}後にはデモプログラムに変わりますが、STOPキーやE RETURN と押して出たメニュー画面^{がめん}は、ファンクションキーを押すまで変わりません。

① BASIC……[F1]

ベーシックでプログラムを作るときに使います。ベーシックについては、ベーシック基礎編をご覧ください。

一度ベーシックモードにしてしまうと、キー入力によりメニュー画面に戻することはできません。ベーシックからメニュー画面に移りたいときには、一度、H1の電源を「切」にし、約5秒後再び「入」にして、メニュー画面を呼び出します。

② モニタ……[F2]

内蔵の簡易機械語モニタを使って、機械語プログラムを作ることができます。内容がかなり難しくなりますから、H1を初めて使う人は、ベーシックなどで、少しコンピュータに慣れてから試してみてください。

間違って、内容がわからないまま[F2]キーを押して、モニタ画面にしてしまったら、英文字の[E]キーを押して「RETURN」キーを押すと、メニュー画面に戻すことができます。

モニタの詳しい使い方は、付録の機械語モニタ仕様をご覧ください。

注) メニュー画面からのモニタの状態で、G「RETURN」とするとメニュー画面になります。

このとき、ファンクションキー[F1]を押すとH1が動作しなくなりますので、電源を一度切り、約5秒後に再び電源を入れてください。

③ ROM カートリッジ……[F3]

画面で、ROM カートリッジの使い方を説明します。説明が終わると自動的にメニュー画面に戻ります。説明の途中でメニュー画面に戻りたいときには、「STOP」キーを押します。

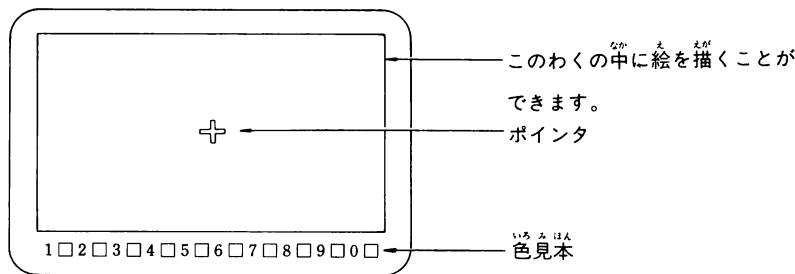
④ スケッチプログラム……[F4]

テレビ画面に、好きな絵を描くことができます。メニュー画面から[F4]キーを押して「スケッチプログラム」の画面を出します。

「スケッチプログラム」の使い方が説明されますので、よく読んでください。

(1) スケッチ用の画面を出しましょう。

「RETURN」キーを押して、スケッチ用の画面を出します。



スケッチ用画面

ポインタ……点を打ったり、線を引いたりするときの位置を決めます。カーソルコントロールキー(←→↑↓)で、動かします。↑+→のように2つのキーを一度に押して、ななめ(ノ)に動かすこともできます。

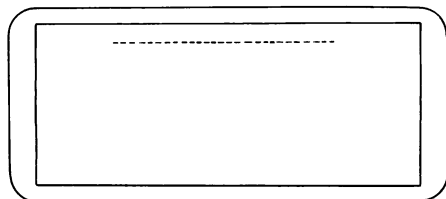
色見本……点や線の色を指定します。数字キーの0～9までを押すと、数字の横に表示されている色で点を打ったり、線を引いたりすることができます。指定した色は、数字の部分が白で反転表示されます。

(2)まず「点」を打ってみよう。

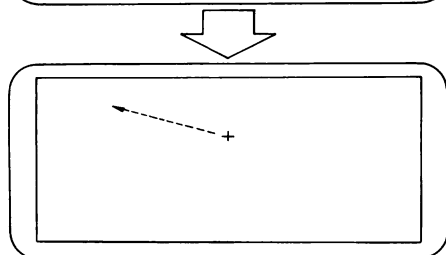
カーソルコントロールキーで、ポイントを「点」を打ちたい位置に動かします。

その位置で「スペース」バーを押せば指定した色で「点」を打つことができます。

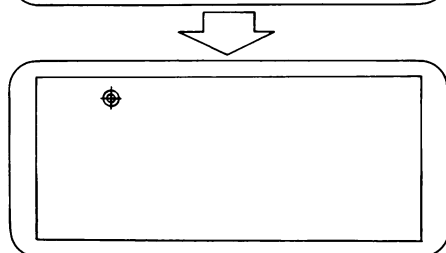
(3)次に「線」を引いてみよう。



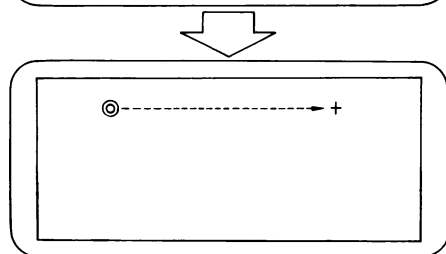
点線の位置に「線」を引くことにします。



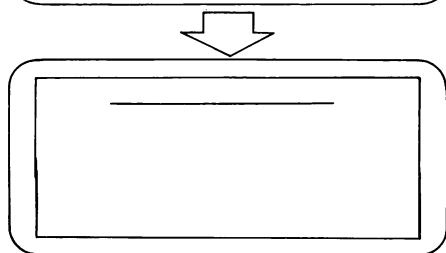
カーソルコントロールキーで、ポイントを引きたい「線」の一方の端まで動かします。



その位置で「L」キーを押すと、◎(赤丸)が表示され、同時に画面の右上に「LINE」と表示されます。



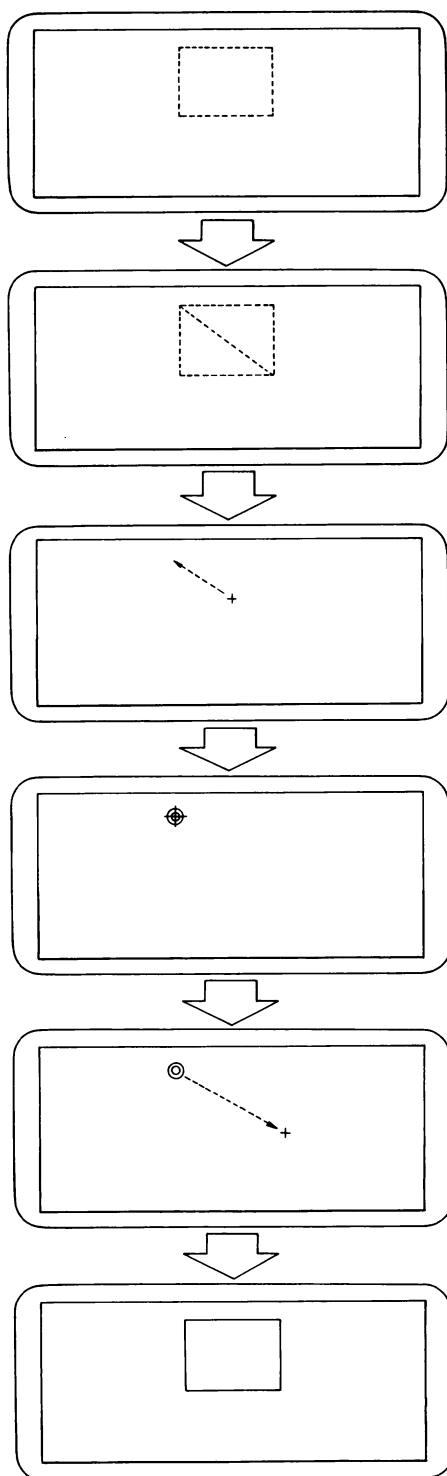
引きたい「線」のもう一方の端までポイントを動かします。



その位置でもう一度「L」キーを押すと、◎(赤丸)と「LINE」の表示は消え、指定した色で「線」が引けます。

また、線はスペースバーを押しながら、カーソルコントロールキーを押すと、押し続けている間、線を引くことができます。

(4)今度は長方形を描いてみよう。



点線のような長方形を描くことにします。

ポインタは、描きたい長方形の対角線上の両端を動かします。

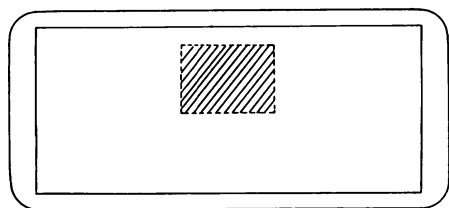
対角線的一方の端まで、ポインタを動かします。

その位置で **[B]** キーを押すと、**◎**(赤丸)が表示され、同時に画面の右上に **(BOX)** と表示されます。

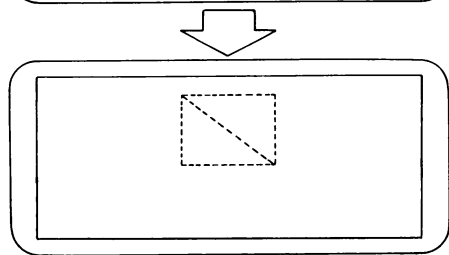
対角線のもう一方の端までポインタを動かします。

その位置でもう一度 **[B]** キーを押すと**◎**(赤丸)と **(BOX)** の表示は消え指定した色で「長方形」が描けます。

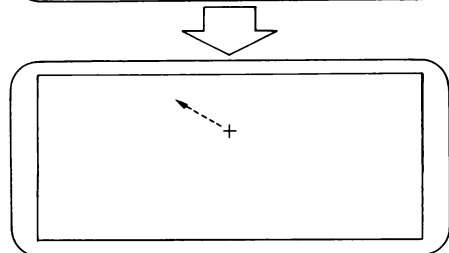
(5) **[F]** キー (FILL) で、長方形の中を塗りつぶそう。



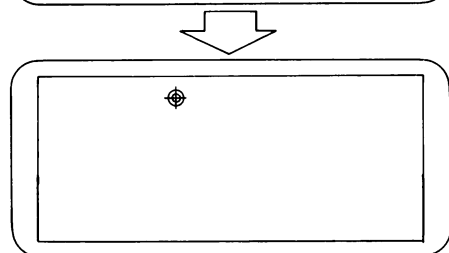
点線の範囲を塗りつぶすことにします。



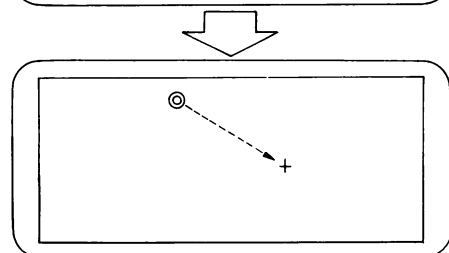
長方形を描くときと同様に、ポインタは、塗りつぶしたい範囲の長方形の対角線の両端を動かします。



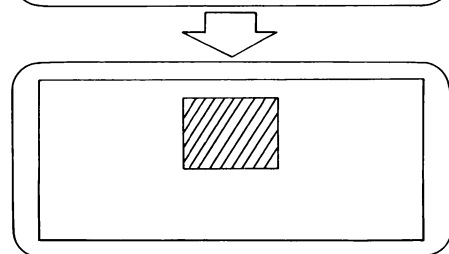
対角線の一方の端まで、ポインタを動かします。



その位置で **[F]** キーを押すと、◎(赤丸)が表示され、同時に画面の右上に (FILL) と表示されます。



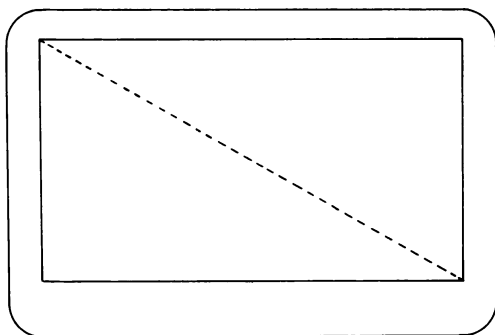
対角線のもう一方の端までポインタを動かします。



その位置でもう一度 **[F]** キーを押すと、◎(赤丸)と (FILL) の表示は消えて指定した色で長方形の中を塗りつぶします。

(6)画面の色を変えてみよう。

長方形の中を塗りつぶしたときと同じように、**[F]** キー (FILL) を使えば簡単に画面の背景を好きな色に変えることができます。



操作方法は長方形の中を塗りつぶしたときと同じです。画面全体を大きな長方形と考えて、今度は画面の対角線の両端に、ポインタを動かせばよいわけです。

(7)終わるときは、

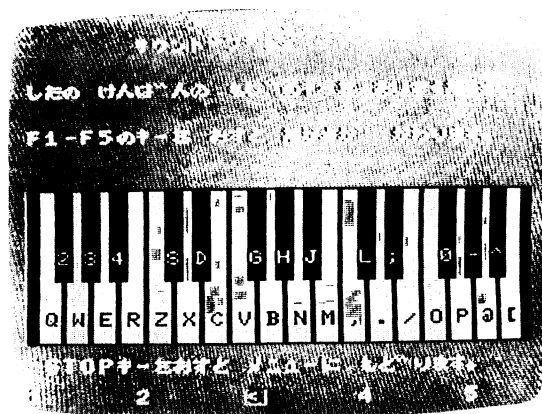
[STOP] キーを押します。
画面はメニュー画面に戻ります。

注) テレビ信号の特性により、たて線と点の色は、指定した色と異なる場合があります。

VDP (ビデオ信号IC) の特性により、画面の横8ドットごとに2色までしか指定できないため、3色目を指定すると、前に指定した色が、変わってしまいますのでご注意ください。

⑤ サウンドプレイ-----[F5]

画面のけんばんにかかっている文字のキーを押して、サウンドプレイを楽しめます。



- (1)けんばんに表示してある文字のキーを押すと、そのけんばんが赤く表示されて、その場所に相当する音が出ます。
- (2)画面の下の1～5の数字は、音の高さをあらわします。最初3に設定されていますが、[F1]～[F5]キーを押して、音の高さを自由に変えられます。数字が大きくなるほど音は高くなります。演奏しているときの音の高さは、数字が赤く囲まれて表示されます。

注) 2つ以上のキーを押しても、単音しか出ません。

終了するときには [STOP] キーを押してください。

画面はメニュー画面に戻ります。

VI おや？ H1が故障かなと思ったら

	症 状	原 因	対 策
テレビ	画面に何も 出てこない。	H1 の電源が入っていない。 テレビとH1の接続が間違っている。 スイッチボックスの切り換えがTV 側になっている。 MB-H1のみ テレビのチャンネルが合っていない。 MB-H1のみ ビデオ用スイッチが入っていない。 スピードコントロールスイッチがス ロー2、スロー3になっている。	電源を“入”にする。 テレビとH1を正しく接続する。 コンピュータ側にする。 テレビのチャンネルを合わせる。 スイッチを“ビデオ”にする スピードコントロールスイッチを ノーマル1にする。
	音が出ない。	テレビの音量調節が最小になってい る。	音量を調節する。
カ セ ッ ト レ コ ー ダ	カセットレコーダが 回らない。	カセットレコーダが一時停止状態に なっている。 カセットレコーダとH1の接続が間 違っている。 操作手順が間違っている。	一時停止を解除する。 正しく接続する。 ベーシック基礎編を読んで正しく 操作する。
	カセットレコーダの プログラムを読み込め ない(ロードできない。)	電池の電圧が下がっている。 カセットレコーダの音量が小さすぎ るか、または大きすぎる。 カセットレコーダの音質調節が合っ ていない。 カセットレコーダとH1の接続が間 違っている。 操作手順が間違っている。 スピードコントロールスイッチがス ロー2、スロー3になっている。	電池を交換する。 音量を調節する。 音質を調節する。 正しく接続する。 ベーシック基礎編を読んで、正し く操作する。 スピードコントロールスイッチを ノーマル1にする。

	症 状	原 因	対 策
カセ ット レ コー ダ	カセットレコーダに プログラムを記録でき ない(セーブできない。)	カセットレコーダの録音レベルが低 すぎる。 カセットテープの誤消去防止用ツメ が折り取られている。 スピードコントロールスイッチがス ロー2、スロー3になっている。	録音レベルを上げる。 テープを交換するか折り取られた部 分をセロハンテープなどでふさぐ。 スピードコントロールスイッチを ノーマル1にする。
H 1 本 体	電源が入らない。	電源コードまたは本体接続ケーブル が接続されていない。 ROM カートリッジ使用の場合、カ ートリッジが正しく差し込まれてい ない。	正しく接続する。 ROM カートリッジの向きを確認 し、奥まできちんと差し込む。
	画面全体に意味不明の 文字や図形があらわれ てキー入力ができない。	H1 が誤動作している。	一度、H1 の電源を切り、5 秒ほ ど待ってから再び電源を入れる。
ジ ョ イ ス テ ィ ツ ク	ジョイスティック が操作できない。	ジョイスティックが正しく差し込ま れていない。 ソフトウェア(プログラム)で選択さ れていない。	正しく接続する。 ゲームカートリッジやカセットテ ープの説明書を読んで、正しく使 用する。

VII H1 の仕様

仕様は、改良のため、予告なく変更することがあります。
本機を使用できるのは、日本国内のみで、外国では使用できません。

●大 き さ 本体：324(幅)×229(奥行)×55(高さ)mm，電源ボックス：330(幅)×93(奥行)×61(高さ)mm
一体化した場合：330(幅)×322(奥行)×61(高さ)mm

●重 さ 本体：MB-H1 1.8kg，MB-H1E 1.7kg
電源ボックス：1.6kg

●使 用 温 度 0～35℃

●C P U Z-80A (相当品)

●メモリ(標準実装) MB-H1：ROM 32Kバイト(ベーシック)+8Kバイト(内蔵ソフト)
RAM 32Kバイト+16Kバイト(V-RAM)

MB-H1E：ROM 32Kバイト(ベーシック)

RAM 16Kバイト+16Kバイト(V-RAM)

●表 示 能 力
画 面 構 成

よこ39文字×たて24行(6×8ドットマトリクス)

よこ29文字×たて24行(8×8ドットマトリクス)電源投入時

文 字 英字(大文字、小文字)、数字、ひらがな、カタカナ、記号、グラフィック文字

グラフィック機能 ハイリゾリューションモード(よこ256ドット×たて192ドット)

マルチカラーモード(よこ256ドット×たて192ドット)

表 示 色 16色(透明、黒、緑、明るい緑、暗い青、明るい青、暗い赤、水色、赤、明るい赤、暗い黄、
明るい黄、暗い緑、紫、灰、白)

●キ ー ボ ー ド JIS規格準拠 73キー(かな文字50音配列)

●プログラミング言語 MSX-BASIC

●サ ウ ン ド 機 能 8オクターブ、3重和音出力

●電 源 AC100V±10%、50/60Hz共用
消費電力 MB-H1 20W

MB-H1E 19W

●カートリッジスロット +5V 300mA(最大) / スロット

コネクタ電源 +12V 50mA(最大)

-12V 50mA(最大)

●ジョイスティック +5V 50mA(最大) / コネクタ

コネクタ電源

VIII アフターサービスと保証

■転居されるときは

ご転居により、お買い求めの販売店のアフターサービスを受けられなくなる場合は、前もって販売店にご相談ください。ご転居先での日立の家電品取扱店を紹介させていただきます。

■保証について

●この商品は保証書付きです。

保証書は、販売店で所定事項を記入してお渡しいたしますので、記載内容をご確認いただき、大切に保存してください。

●保証期間は、お買い上げの日から1年間です。

なお、保証期間中でも有料になることがありますので、保証書をよくお読みください。

●保証期間経過後の修理については、販売店にご相談ください。

修理によって機能が維持できる場合は、お客様のご要望により有料修理いたします。
当社は、販売店からの注文により、補修用性能部品を販売店に供給します。

■補修用性能部品の保有期間について

当社は、このパーソナルコンピュータの補修用性能部品を、製造打切後最低6年間保有しています。

■アフターサービスなどでお困りの場合は

アフターサービスについてご不明の場合、その他お困りの場合は、お買い上げの販売店か別紙「ご相談窓口一覧表」のご相談窓口にお問い合わせください。

— お客様メモ：サービスを依頼されるとき、お役にたちます。

購入店名： _____ 電話 _____

ご購入年月日：昭和 _____ 年 _____ 月 _____ 日

59-01-06
004 責 0.001 100
428 部 ★32000
買 1 個 ★32000 現計
★35000 預り
★3000 釣
7356 213-13 円

ベーシック基礎編

皆さんがH 1に何かをさせようとするときには、コンピュータ用の言葉を使ってH 1に命令しなければなりません。そのコンピュータ用のことばがベーシックなのです。

ベーシック基礎編では、H 1に命令するときの言葉や、命令の仕方を説明します。最初から順に一つ一つ実際に、H 1を使いながら練習していきましょう。基礎編が終わるころには、きっとプログラムが作れるようになりますよ。

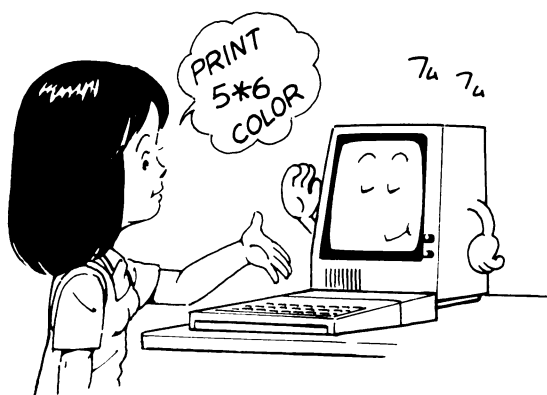
I とにかく使^{つか}ってみよう

ベーシックって何？ …まずはじめに

私たちが話をするためには言葉が必要です。コンピュータも同じなのです。人間がコンピュータを使う場合、お互いに通じ合う共通の言葉が必要です。このような言葉を「コンピュータ言語」と言います。

人間の使う言葉にも日本語、英語、ドイツ語などたくさん種類があるように、コンピュータ言語にも用途に応じていろいろなものがあります。H1が持っているコンピュータ言語は「ベーシック」という言葉で、覚えやすく、いろいろな目的に使えるのでほとんどのパーソナル・コンピュータが持っているものです。

ベーシックはアメリカ生まれの言葉ですからちょっと英語に似ていますが、覚えるのはとても簡単です。さあ、H1のベーシックをこれから学んでいきましょう。



ベーシック ビギナーズ オールパーパス シンボリック
BASIC = Beginner's Allpurpose Symbolic
Instruction Code



テレビとH1の電源を入れます。

メニュー画面が出てきましたね。

さあ、**[F1]** キーを押して、ベーシックのスタートです。

画面に何か書いてみよう

…一番やさしい PRINT 命令

MB-H1

```
MSX BASIC version 1.0
Copyright 1983 by Microsoft
28815 Bytes free
Ok
■ ←———カーソル
```

MB-H1E

```
MSX BASIC version 1.0
Copyright 1983 by Microsoft
12431 Bytes free
Ok
■ ←———カーソル
```

テレビとH1の電源を入れメニュー画面の状態です。F1キーを押すと画面は図のようになっていますね。(MB-H1Eの場合は、テレビとH1の電源を入れるだけで図のような画面になります。) この表示をスターティング・メッセージといい、H1でベーシックが動きはじめて、いつでも命令をうけつけられる状態になったということの意味をしています。字のすぐ下にある白い四角形「■」をカーソルといいます。

キーボードからいろいろなキーを押してみましょう。押したキーの通りに画面がでますね。見やすくするために **CAPS LOCK** キーを押して英字の大文字にしましょう。そして、キーを押して

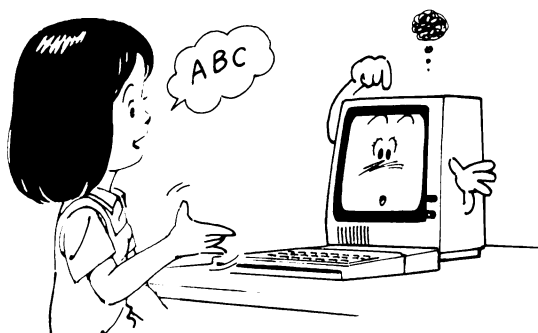
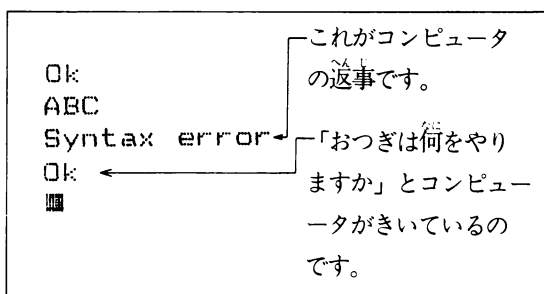
ABC

としてみてください。画面には「ABC」と出るだけで、その横にはカーソルがあるだけで他には何も変わったことはないようです。「ABC」は画面に出ている

だけで、コンピュータにはまだ伝わっていないのです。さあ「ABC」をコンピュータに伝えてみましょう。画面に出ている文字をコンピュータに伝えるときには、**RETURN** キーを押します。

ABC **RETURN**

RETURN 記号は今後 **RETURN** キーを押すことを意味するものとします。何度も出てくるので忘れないでください。



さて、H1は何か返事をしてくれましたか？あれっ、ピッという音がして画面に何か出てきましたね。

Syntax error

これは「あなたのいうこと（キーボードから打ち込んだこと）はわかりません」とH1が返事をして
いるのです。「ABC」とは人間にしか通じない言葉で
すから、H1は理解することができません。だから、
H1にわかる言葉で、あなたの言いたいことを伝えな
ければなりません。コンピュータは決まった形で命令
をしてやらないと、全々受けつけてくれないのです。
ピツという音とともに出てくる表示を、エラーメッセ
ージといい、あなたの言いたいことがわからないとか、
その通りにすることはできないというH1からの返事
なのです。Syntax errorの他にまだたくさんのエラ
ーメッセージがありますが、それらはそのたびごとに
説明していくことにします。

●PRINT(プリント)

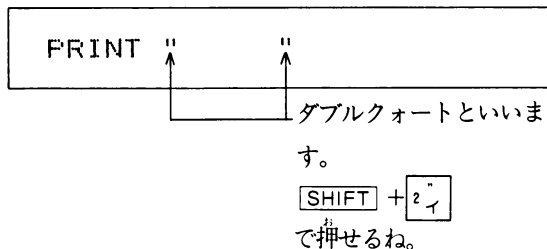
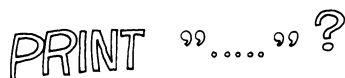
それでは、H 1 に返事をさせるために一番簡単な命令から覚えていくことにしましょう。それは PRINT 命令といいます。

```
PRINT "オハヨウ" RETURN
```

と打ち込んでみましょう。最後に **RETURN** キーを
忘れずに押してください。

カタカナを出すときは、一度 かな キーを押してからでしたね。

```
PRINT"オハヨウ"  
オハヨウ  
Ok  
■
```



おや？今打ち込んだ文字のすぐ下に“オハヨウ”と出
ました。これがH1の返事なのです。PRINTは、「画
面に書きなさい」という意味の命令なのです。この
PRINTは基本的な命令なので、この他にもいろい
ろな使いかたがあります。少しずつ説明していくことに
しましょう。

```
PRINT 2+3 RETURN
```

と打ち込んでみましょう。

2 + 3の答、つまり5という数字が画面に出てきますね。PRINT 命令のあとに計算式を書いておくと、その答を画面に出してくれます。次に、

```
PRINT "2+3"
```

としてみましよう。

今度は2 + 3がそのまま画面に出てきました。「”」(ダブルクォート) でかこまれたものを **PRINT** 命令で書くと、それをそのまま画面に出してくれます。**PRINT** 命令を使っていろいろなものを画面に出してみましょ
う。

PRINT "アオヤマ"

PRINT 50-30

PRINT "A B C"

最後に画面にH1と書いてみましょう。

```
PRINT H1
○
```

おや、変ですね。なぜ0になるのかは、後で説明することになります。ここでは、ダブルクォートでかこんで

```
PRINT "H1"
```

とします。

●PRINTの省略形

PRINTと書くかわりに省略形「?」（疑問符）を使うことができます。

```
? 2 + 3 RETURN
```

とするとききほどと同じように答えの5が出ますね。

●間違いの直し方

あなたはキーボードを使うのは初めてですか？もしそうでしたら、キーを押すときに、よく押し間違いをするでしょう。

ここでは、押し間違えたときのなおし方を説明しましょう。

イ) 文字の訂正

「PRINT」と打ちたかったのに「ORINT」と打ってしまった。

```
ORINT ■
```

カーソルキー \leftarrow でカーソルを「O」のところまでもっていきます。

```
ORINT
```

ここで「P」を押します。

```
PRINT
```

あとはカーソルを \leftarrow でもとの位置にもどして続きを書きます。

ロ) 文字の削除

「PRINT "オハヨウ"」と打ちたかったのに、PRINT "オハハヨウ"と打ってしまった。

```
PRINT "オハハヨウ" ■
```

\leftarrow キーでカーソルを「ハ」のところまでもっていきます。

```
PRINT "オハハヨウ"
```

ここでDELキーを押すとカーソルの「ハ」が消えて、カーソルより右に並んでいた文字が1字ぶん左によります。

```
PRINT "オハヨウ"
```

カーソルをもとに戻してもいいし、1行にこれ以上書かないなら \leftarrow RETURNキーを押します。

ハ) 文字の挿入

「PRINT」と打つところを「PRNT」と打ってしまった。

```
PRNT ■
```

\leftarrow キーでカーソルを「N」のところまでもっていきます。

```
PRNT
```

ここでINSキーを押すとカーソルが下半分の大きさに表示されます。

ここで「I」を押します。

```
PRINT
```

もう一度INSキーを押すか、 \leftarrow キーを押して、カーソルがもとの大きさにするようにします。（ \leftarrow RETURNキーを押したときもカーソルがもとの大きになります。）

計算だってできるんだ

…たし算、ひき算、かけ算、わり算

PRINT 命令を使って、H1を電卓と同じように使うこともできます。

PRINT 計算式

PRINT 計算式は、計算式の結果を画面に書かせなさいという命令なのです。

●たし算、ひき算

たし算、ひき算は普通のとおりには書けば計算してくれます。

PRINT 5+4 RETURN

PRINT 70-52 RETURN

このとおりに打ち込んでみましょう。ちゃんと計算してくれますね。次に、

PRINT 0.1+5.5-2.3 RETURN

としてみましょう。たし算、ひき算の混じった計算も、小数点の計算もしっかり答えがでますね。

```
PRINT 5+4
9
Ok
PRINT 70-52
18
Ok
PRINT 0.1+5.5-2.3
3.3
Ok
■
```

●かけ算、わり算

かけ算とわり算は、普通とはちょっと違った記号を使います。

かけ算の記号…* アスタリスクと読みます
わり算の記号…/ スラントと読みます

それでは計算してみましょう。

PRINT 5*2 RETURN

PRINT 1/3 RETURN

PRINT 10*7/2 RETURN


```
Ok
PRINT 5*2
10
Ok
PRINT 1/3
.3333333333333333
Ok
PRINT 10*7/2
35
Ok
■
```

●計算の優先順序

$2 + 3 \times 5$ は、25ではなく17だということはわかりますね。かけ算、わり算はたし算、ひき算よりも先に行うのでしたね。こういう規則を「計算の優先順序」といいます。

ベーシックの場合は、次のように決まっています。

$$() \rightarrow ^ \rightarrow \left\{ \begin{array}{c} * \\ / \end{array} \right\} \rightarrow \left\{ \begin{array}{c} + \\ - \end{array} \right\}$$

次の命令文は何を計算しているかわかりますか？

```
PRINT 1+2^4*5^(1+2)/2 RETURN
```

普通の計算式にすると、 $1 + 2^4 \times 5^3 \div 2$ となります。答は1001ですよ。

```
Ok
PRINT 1+2^4*5^(1+2)/2
1001
Ok
■
```

●かっこつきの計算、べき乗の計算

計算式には かっこを使うことができます。

```
PRINT ((5+4)*12+4) / 2 RETURN
```

「(」、「)」はいくつ重ねて使ってもかまいません。ただし「(」の数と「)」の数が合っていないと Syntax error になります。

べき乗の計算もできます。 2^{10} を計算したければ

```
PRINT 2^10 RETURN
```

としてください。

べき乗の記号...^ (指数記号といいます。)

```
PRINT ((5+4)*12+4) / 2
56
Ok
PRINT 2^10
1024
Ok
■
```

●数の表示

次のように打ち込んでください。


```
PRINT 100000*100000*100000 RETURN
```

```
Ok
PRINT 100000*100000*100000
1E+15
Ok
■
```

おやおや？ $1 \text{ E} + 15$ なんて、おかしい表示が出てきましたね。これは1のあとに0が15個並ぶという意味なのです。

```
PRINT 1/(100000*100000) RETURN
```

としてください。

```
Ok
PRINT 1/(100000*100000)
1E-10
Ok

```

$1 \text{ E} - 10$ と出ましたね。これは1の左に0が10個並ぶ（つまり0.0000000001）ということを示しているのです。

このように、非常に大きな数や非常に小さな数は「E」を使った形で画面に表示されることになります。

●〈起こりやすいエラー〉

● Missing operand(ミッシング オペランド)

かけ算、わり算などで、必要な数を書いていないときに起こります。

```
PRINT 3* RETURN
```

とした場合などです。「*」のあとになにか数が必要ですね。

● Division by zero(ディビジョン バイ ゼロ)

0でわり算した場合に起こります。

● Overflow(オーバーフロー)

計算の結果が大きくなりすぎた場合に起こります。エラーが起きない最大の数は

$9.9 \cdots 9 \text{ E} + 62$

14ケタ

負の数も

$-9.9 \cdots 9 \text{ E} + 62$

14ケタ

Ⅱ プログラムの^き基^そ礎

プログラムって何？

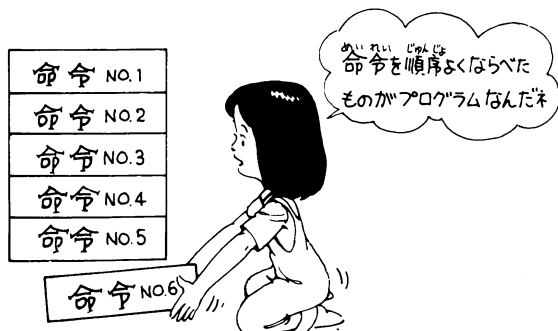
…プログラムを作る前に

クリアスクリーン
(CLS)

これまで、H1に何かさせる場合、まず命令を書いて次に文字や計算式などを書き、最後に「RETURN」とすればよかったのでしたね。しかし、もっと複雑なことをコンピュータにさせたい場合、これでは大変です。

何をどういう順序で実行すればよいのかが、あらかじめわかっているのなら、そのとおりの順序で命令を覚えさせておけばよいと思いませんか？ 命令を全部覚えさせたあと「実行しなさい」という命令をH1に与えてやれば、1回の指示でコンピュータに連続した仕事をさせることができるわけです。

これが「プログラム」の考え方です。つまりプログラムとは、命令を順序よく並べたものなのです。もちろん、命令はコンピュータが理解できるベーシック言語を使います。

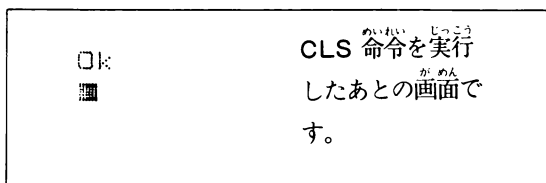


●CLS (クリアスクリーン)

さて、プログラムを作る前に、気分を一新するため、画面をきれいに消してみよう。そんなとき使うのがCLS命令です。

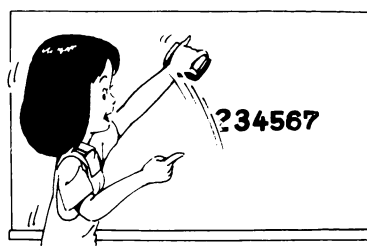
CLS 「RETURN」

と打ち込んでください。画面に書かれている文字や記号がすべて消えて「Ok」とカーソルだけが左上にありますね。



「SHIFT」キーを押しながら「CLS HOME」キーを押しても同じ結果になります。「Ok」は出ませんが、画面はきれいになりましたね。

CLS 「RETURN」
(「SHIFT」 + 「CLS HOME」)



コンピュータは変数が好き …変数、代入文の使い方

PRINT 文で計算できることは説明しました。ここまでは電卓と同じです。コンピュータでは電卓と違う方法で計算することもできます。それは、コンピュータが好きな変数というものを使うのです。

```
A = 5 RETURN
PRINT A RETURN
```

と打ってみましょう。ちゃんと5という答がでますね。

```
Ok
A=5
Ok
PRINT A
5
Ok
■
```

どうやら、ここで使っている「A」は数を覚えているようですね。この「A」が変数と呼ばれているものなのです。変数とは数を入れるための箱があり、その箱につけられた名前であると考えてください。

今度は次のように打ち込んでみましょう。

```
A = 5 RETURN
B = A * 10 RETURN
PRINT B RETURN
```

```
A=5
Ok
B=A*10
Ok
PRINT B
50
Ok
■
```

答は50ですね。

このように、変数は計算式の中に使うこともできます。ですから、

```
A = 10 RETURN
A = A + 1 RETURN
```

と書くこともできます。A=A+1に注意してください。「AはA+1に等しい」ではありませんよ。「=」は「右側のものを左側に入れる」の意味です。

ですから、A=A+1は「Aに1を加えたものを、もう一度Aの箱に入れる」という意味になります。実際にそうなっているかどうか確かめてみましょう。

```
PRINT A RETURN
```

10に1を加えた11になっていますね。

```
A=10
Ok
A=A+1
Ok
PRINT A
11
Ok
■
```

変数名 = 式

PRINT 変数名

「=」(イコール)は「等しいの意味」
じゃなくて「右側のものを左側の箱に
入れなさい」という意味になります。

変数の名前にはAやBやXなどいろいろなものが
使えます。それについては後で説明します。



A = B



A = 3



●変数名に使える文字は

さっきの例では、変数名にAやBを使いましたが、も
っといろいろな種類の文字を使うことができます。
ただし、次の3つの規則を守ってください。



PRINT=3

変数名の3つの規則

①先頭は英字（アルファベット）で、その後には英字か、数字を使うこと。

例) A, B, ABC, A1, X20, X3Y……変数

2AB……先頭が英字でないので変数でない。

A¥7……「¥」という数字でも英字でもない文字を使っているの
で変数でない。

②長さは255文字まで。ただし、3文字以上の部分は変数名の区別には使われない。

例) A123456787 } すべて変数だが同じものと
A123456788 } みなされる。
A12345678B }

最初の2文字だけが区別に使われる。

③ベーシックですでに決められている言葉を使ってはいけない。

例) PRINT1……PRINT命令とみなされて変数にはならない。

XPRINT……やはり変数には使えない。

PR, XPR……変数として使える。

次のように打ち込んでみましょう。

SEIKO=10 RETURN

HIROMI=20 RETURN

PRINT SEIKO+HIROMI

答は30と出ましたか

それでは変数SEIKOの内容を確かめてみましょう。

PRINT SEIKO RETURN

10となっていますね。

いろいろな変数を作って計算させてみましょう。コン
ピュータは変数が大好きですから、これからたくさん
変数が出てきます。変数と定数(決まった数字)の計
算などをなれるまで練習してください。

ところで、PRINT命令で

PRINT H1 RETURN

としたら、0と表示されましたね。もうおわかりです
ね。H1が変数として計算されたのです。変数H1に
は何も入れていませんから、変数H1の中はからっぽ、
つまり0と答えてくれたのです。もちろんほんものの
H1は、からっぽではありませんよ。

ぎょう ばん ごう じゅん じょ 行番号で順序よく

ぎょう ばん ごう かんが かた …行番号の考え方

ニュー エンド ラン
(NEW, END, RUN)

さあ、やっとプログラムを作るところまでやってきました。その前に、必ずしておかなければならない命令 NEW があります。

- NEW(ニュー)……コンピュータの頭をからっぽにする。
プログラムを作り始める前に必ず

NEW RETURN

としてください。プログラムは、コンピュータに順序よく命令を覚えさせるものでしたね。それで、新しくプログラムを作るときには、前に覚えておいた命令を全部消しておかないと、古いプログラムと新しいプログラムがごっちゃになってしまうのです。

NEW RETURN とすることでコンピュータのなかみはからっぽの状態になります。(電源を入れたときと同じ状態。)

これからは、たくさんの例題プログラムがでできます。そのたびに NEW 命令を実行して、毎回きれいな状態でコンピュータを使えるようにしておいてください。

● 行番号って何？

さあいよいよプログラム作りです。まずはプログラムに絶対に必要な行番号のお話から。

命令が順序よく並べられたものがプログラムだと言いましたね。さて、その順序とはどうやって決めるのでしょうか？ベーシックでは、命令の先頭に番号をつけることで順番を決めています。たとえば次のように打ち込んでください。

```
10  A = 1   RETURN
20  B = 5   RETURN
30  C = A + B   RETURN
40  PRINT  A   RETURN
50  PRINT  B   RETURN
60  PRINT  C   RETURN
70  END   RETURN
```

あれ？なんだかいつもと違いますね。 RETURN としても「Ok」が出てきませんし、PRINT 命令の結果も表示されていません。

原因は、先頭に書かれた10～70の数字にあるのです。

1 行の最初に数字が書かれると、コンピュータはこの命令を実行せずに、数字の順番を覚えておくのです。

このように先頭に番号のついた命令文の集りを、プログラムといいます。

行番号 命令 RETURN

行番号には0~65529までの
整数が使えるよ

1つの行の命令の最後には
かならず RETURN キー
を押すこと

こういうふうに打ち込むとコンピュータは
命令を覚えていくんだね



この、先頭に書く数字を行番号といいます。コンピュータは「実行しなさい」という命令を受けたときに、行番号の小さい順から順序よく実行していきます。

●END(エンド)………プログラムの終わり
行番号70に見かけない言葉がありますね。これをEND命令といいます。「ここまできたらプログラムの実行をやめなさい。」という意味をもつ命令です。

●RUN(ラン)………プログラムのスタート
さあ、プログラムを実行してみましょう。
プログラムを実行させる命令はRUN命令です。次のように打ち込んでください。

RUN RETURN

画面には3つの数字が表示されて「Ok」が出ました。
どうやら、ちゃんと計算してくれたようですね。

```
Ok
10 A=1
20 B=5
30 C=A+B
40 PRINT A
50 PRINT B
60 PRINT C
70 END
RUN
1 ←——— これが行番号40の結果
5 ←——— " 50 "
6 ←——— " 60 "
Ok
■
```

●うまく動かなかった人のために

「ピッ」という音がして
シンタックス エラー イン
Syntax error in ○○

や
ミッシング オペランド イン
Missing operand in ○○

などというエラー・メッセージが出た人はもう一度よく画面とプログラムを比較してみましょう。エラー・メッセージの最後の数字がエラーの起こった行番号です。その行のどこかに間違った文字を打っていませんか？その場合には、間違った場所にカーソルを動かして文字を書き直しましょう。書き直しが終わったら、もう一度RUNしてください。今度はうまくいきましたか？もっと詳しい方法は次に説明します。（打ち間違いのなおし方は73ページで説明しましたね。

〈よく起こるエラー〉

●Syntax error (シンタックス エラー)
ベーシックにない命令を使うとこのエラーが起きます。
たいていは、キーの押し間違いが原因です。

●Missing operand (ミッシング オペランド)
315ページを見てください。

プログラムを書き換えよう

…プログラムの修正

リスト
(LIST)

●LIST(リスト)……プログラムを画面に表示させる。(LISTすると言います。)

83ページのプログラムはうまく動きましたか？こんどは、**SHIFT** + **CLS HOME** キーを押して、画面を消してしまいましょう。あれっ？せっかく作ったプログラムが見えなくなっていましたね。でも安心してください。画面から消えてしまってもコンピュータはちゃんと覚えているのです。

LIST **RETURN**

と打ち込んでみましょう。さっき打ち込んだプログラムを全部画面に書いてくれます。ファンクションキー **F4** で一度に入力することもできます。

```
Ok
LIST
10 A=1
20 B=5
30 C=A+B
40 PRINT A
50 PRINT B
60 PRINT C
70 END
Ok
■
```

プログラムの一部を見るときには

LIST 始めの行番号 - 終わりの行番号

「-」はマイナス記号を使う。
(「-」は数字キーのとなりの **＝** のキーですよ。)

LISTの書きかたあれこれ

LIST ……プログラム全部を画面に表示する
LIST 20 - ……行番号20から最後まで画面に表示する
LIST -50 ……最初から行番号50まで画面に表示する
LIST 30 - 60 ……行番号30から60まで画面に表示する

●ちょっと修正してみよう

イ) 命令を書き換えたいときは…
行番号30の $C = A + B$ を

$C = A * B$

に換えたい場合、どうしたらよいでしょうか？方法は簡単です。**⇨**キーや **⇩**キーを使って行番号30までカーソルを移動し、「+」を「*」に換えればよいのです。換えた後に **RETURN** とするのを忘れてはいけません。変更したあと **RETURN** キーを押すとコンピュータは命令を受けつけるのでしたね。

ロ) 命令を1行分消したいとき…
今度は、行番号40と50の命令文を消してみましょう。

```
40 RETURN
50 RETURN
```

としてください。行番号だけ押してすぐ RETURN
とすると、コンピュータは、その行番号の命令は忘れ
てしまうのです。LISTして確認してください。

```
Ok
40
50
LIST
10 A=1
20 B=5
30 C=A*B ← 行番号40、50は
60 PRINT C   なくなった。
70 END
Ok
■
```

```
Ok
LIST
10 A=1
20 B=5
30 C=A+B
40 PRINT A
50 PRINT B
60 PRINT C
70 END
Ok
■
```

☐、☐ キーでカーソルをここまで移動する。

また、別の方法もあります。改めて

```
30 C = A * B RETURN
```

と打ってやるのです。

コンピュータは、同じ行番号の命令文が二つあれば、
あとから入れた方を覚えてくれるのです。それでは

```
LIST RETURN
```

として修正されているかどうか見てみましょう。ちゃんと書き換わっていますね。

```
Ok
30 C=A*B
LIST
10 A=1
20 B=5
30 C=A*B
40 PRINT A
50 PRINT B
60 PRINT C
70 END
Ok
■
```

ハ) 新しい命令文をつけ加えたいときは…
今度は、行番号60と70の間に

PRINT A/B

という命令を追加したい場合、どうしたらよいでしょうか。60と70の間に実行されるのですから、

65 PRINT A/B RETURN

としてやればよいのです。(61~69の間の整数なら、どれでもかまいません。) さあ LIST してみましょう。

```
Ok
65 PRINT A/B
LIST
10 A=1
20 B=5
30 C=A*B
60 PRINT C
65 PRINT A/B ← ちゃんと60と70の
70 END          間にはいつていま
Ok              す。
■
```

このように、プログラムには新しい命令文を付け加えることがあるので、行番号は10番飛びにつけておくのが便利です。もし行番号を1から順番にぎっしりつめて書いておいたら、今のように、新しい命令文を付け加えることはできませんね。

プログラムを保存するには …セーブ、ロードの方法

シーセーブ シーロード シーロードベリファイ
(CSAVE, CLOAD, CLOAD?)

H1ではプログラムをカセットテープに保存（記録）しておくことができます。また必要なときにもう一度H1の中に読み込むこともできます。このときに使うのがCSAVE, CLOAD, CLOAD?命令です。

●CSAVE(シーセーブ)

プログラムをカセットに記録する（これをセーブと言います）命令がCSAVEです。

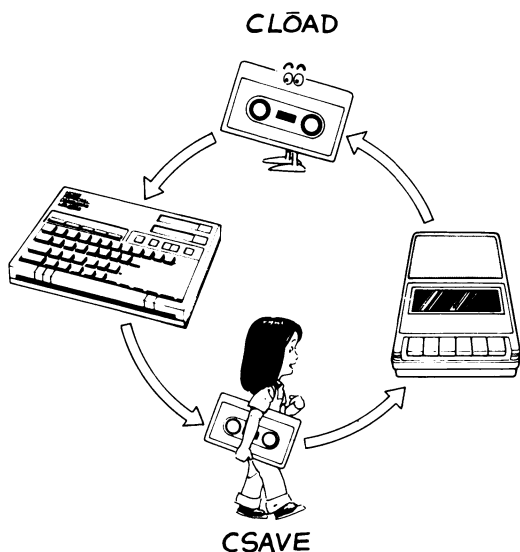
CSAVE “適当な名前”

「適当な名前」は、記録されたプログラムを、名前で区別するために使います。6文字以内で「」を含まなければ、どんな文字を使ってもかまいません。なるべく、プログラムの内容が思いだせるような名前を使いましょう。たとえば、記録したいプログラムが、テニスゲームであれば、

CSAVE “テニス”

とかするわけです。

例として、86ページのプログラムをセーブしてみましよう。プログラムの名前を“TEST”としてみます。



プログラムをCSAVE, CLOADする前に、カセットレコーダがH1に正しく接続されているかどうか確かめてください。接続の方法は取扱編の30ページをよく読んでください。

(1) まず、カセットテープを用意して、カセットレコーダに入れます。カセットテープは、一般の音楽用テープでも使えますが、なるべくプログラム用の短いテープを使いましょう。

音楽用のC-90やC-120などの長時間記録用のテープを使うと、長すぎてテープのどこにプログラムを入れたのかさがすのが大変ですよ。

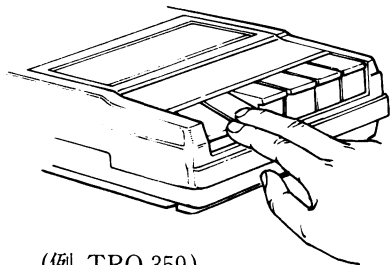


プログラム用のカセットテープを用意してね!!

(2) 録音レベル調節つまみ（ボリューム）とトーンコントロールを調節します。（取扱編31ページをご覧ください。）TRQ-359の場合は自動的に適正なレベルで記録されますので調節の必要はありません。

(3) カセットレコーダの録音ボタンと再生ボタンをいっしょに押してください。レコーダにリモート端子がある場合はカセットレコーダは止まったままですが、リモート端子のない場合は動き出します。

（ただし、カセットテープへの記録は始めません。）



（例 TRQ-359）

(4) **CSAVE"TEST"** と打ち込んでください。

(5) **RETURN** キーを押します。

リモート端子がついているレコーダの場合、ここで初めてテープが回り始め、プログラムが記録されます。このとき、テープカウンタの数字をメモしておくとCLOAD（シーロード）やCLOAD?（シーロードベリファイ）するときの頭出しに便利です。

へ）しばらくテープが回ったあと、画面に「Ok」が表示され、カーソルが出てきます。これでプログラムはテープに記録されました。

CSAVE"TEST"
Ok ←

Okが出るまでの時間は、プログラムの長さによって違います。

リモート端子がついているカセットレコーダはセーブが終ると自動的にテープが止まります。そうでないレコーダは、テープが回り続けます。いずれの場合も停止ボタンを押して停止状態にしてください。

以上で、セーブは終わりです。テープに録音してもプログラムはH1の中に残っています。LISTして確認してください。ちゃんと記録されたかどうか心配ですか？それを確かめるために、**CLOAD?** 命令があります。

●CLOAD? (シーロードベリファイ)

CLOAD? 命令は、プログラムがカセットテープの中にちゃんと記録されたかどうかを調べる (ベリファイすると言います) 命令です。

●**CSAVE**を行なったあとには、必ずこの**CLOAD?**を行なって記録がうまくいったことを確認してください。

CLOAD? "プログラム名"

「プログラム名」には、ベリファイしたいプログラムの名前を書きます。そのプログラムが見つかるまで、テープは回り続けるので注意してください。

また、「プログラム名」を省略することもできます。その場合、一番最初に見つけたプログラムをベリファイします。

さっきセーブしたプログラムをベリファイしてみましょう。

(1) プログラムを記録してあるカセットテープをカセットレコーダに入れます。記録のときメモしておいたテープカウンタの数を参考にしてプログラムがはいっている場所まで (TRQ-359 の場合にはリモート端子 (REM) に接続したプラグを一時はずして行なうか MOTOR ON/OFF 命令を実行して) 巻きもどしてください。

(2) **CLOAD?** "TEST" と打ち込んでください。

(3) カセットレコーダの再生ボタンを押し、

RETURN を押します。リモート端子がついている場合と、そうでない場合の違いは、**CSAVE** のときと同じです。プログラムが見つかったら「Found: TEST」と表示されカセットテープ (位置出しが終わったらリモート端子の接続をもとどおりにします。) に記録されたプログラムと、H1 の中のプログラムとの比較が始まります。

二) しばらくテープが回ったあと、画面に「Ok」と表示されればプログラムは、正確に記録されていたことになります。「Verify error」と表示された場合は、正確な記録はされていません。もう一度セーブをやりなおしてください。

CLOAD? "TEST"

Found: TEST

Ok



正確に記録されていた場合

CLOAD? "TEST"

Found: TEST

Verify error

Ok



正確に記録されていなかった場合

「Ok」が出れば、ベリファイは終了です。あとは H1 の電源を切ったり、NEW 命令を実行したりして、プログラムを消してしまっても大丈夫です。ちゃんとカセットテープに記録してあるのですから。

●CLOAD (シーロード)

CLOAD 命令は、カセットテープに記録しておいたプログラムを、H1 の中に再び読み込む (ロードすると言います) 命令です。プログラムを読み込んだ場合、そのとき H1 の中にはいていたプログラムは消えてしまうので注意してください。

CLOAD "プログラム名"

プログラムを読み込むという点をのぞいて、あとは **CLOAD?** 命令の操作と同じです。

今度は記録してあるプログラムをロードしてみましょう。まず

NEW RETURN

としてH1の中に残っていたプログラムをすべて消してしまいましょう。NEW しなくともロードはできますが、ここでは、ほんとうに読み込んだことをテストするために、NEW をしてみます。
ベリファイのときと同じ手順で、カセットを操作してください。キーボードからは、

CLOAD "TEST"

と打ち込みます。「Ok」が出れば、読み込み完了です。プログラムはH1の中に入りましたか？LIST してみてください。

```
NEW
CLOAD"TEST"
Found: TEST
Ok
LIST
10 A=1
20 B=5
30 C=A*B
60 PRINT C
65 PRINT A/B
70 END
Ok
■
```

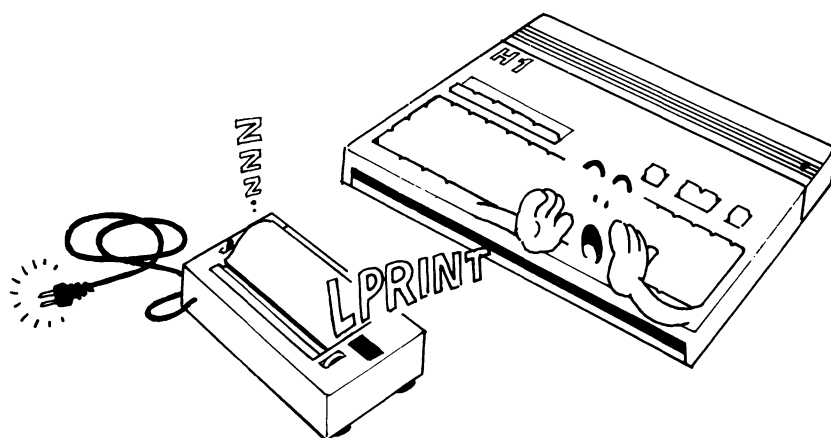
注) 以上の命令を実行しても画面に何も返事が戻ってこない場合は CTRL キーを押しながら STOP キーを押して、もう一度最初からよく読んで、やり直して下さい。また、取扱編30ページ「カセットレコードとの接続」もご覧ください。

つか プリンタを使おう

エルプリント エルリスト (LPRINT, LLIST)

画面にプログラムを表示させたいときは、LIST 命令でしたね。でも、せっかく作ったプログラムを紙に打ち出すと、前後関係がよくわかって便利ですね。そういうときに、作ったプログラムをプリンタに打ち出す命令が LLIST なのです。画面に何かを表示させたいときには、PRINT 命令でしたが、プリンタに何かを表示

させたいときには、LPRINT 命令です。PRINT 命令や LIST 命令と同じ使い方をします。ただし、LLIST や LPRINT 命令を実行するときには、プリンタが正しく接続されているかどうか、用紙が正しくセットされているかどうか、電源が入っているかどうかを確認してください。



もし間違えて、プリンタの電源が「切」のときに LLIST などとしてしまったら、H1 は、いつまでも命令を実行しようとして待ちつづけますので、[CTRL] キーと [STOP] キーを同時に押して、もとのコマンド待ちの状態にします。もしプリンタが接続されてい

なかったときには、作ったプログラムを一度カセットテープに記録(CSAVE を使うのでしたね)してから、H1 の電源を切って、接続します。接続が終ったら電源を入れ、カセットテープからプログラムをロードしてから、プリンタにプログラムを打ち出します。

III ベーシックを勉強しよう

はじめに

オート リナンバー デリート リマーク (AUTO, RENUM, DELETE, REM)

この章から、いよいよ本格的に、H1のベーシックの命令を説明していきます。

まずは、覚えておく便利な4つの命令から。

●AUTO(オート)

プログラムを作るには、どうしても行番号を先頭につけなければなりません。プログラムが長くなったりするといちいち打ち込むのは大変手数がかかります。そんなときにこのAUTO命令を使うと便利です。AUTOは行番号を自動的に画面に書いてくれる命令なのです。

```
AUTO 100, 10 RETURN
```

としてみましょう。はじめの行番号として100を表示してくれました。それではその行番号のところに

```
100 A=10 RETURN
```

と書いて下さい。次の行に110という行番号を書い
てくれましたね。どうやら10ずつ増える行番号を自動的に
書いてくれるようです。どんどん命令を打ち込んで
みましょう。

```
110 B=A*10+3 RETURN
```

```
120 PRINT B RETURN
```

```
130 END RETURN
```

```
AUTO 100,10
100 A=10
110 B=A*10+3
120 PRINT B
130 END
140 <
```

ここで **CTRL** キー
を押しながら
STOP キーを押
す。

このようにして、プログラムを打ち込み終わったら
CTRL キーを押しながら **STOP** キーを押すと、行
番号の自動発生が止まります。

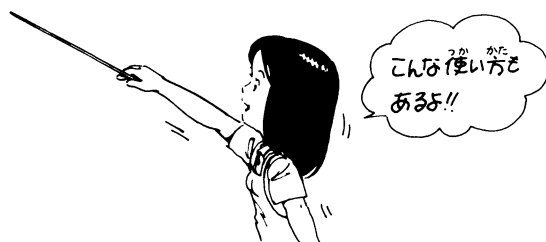
AUTO はじめの行番号, 行番号の間隔

はじめに行番号と行番号の間隔を省略してAUTO
RETURN とすると、行番号10から10おきに書いてく
れます。これから出てくる例題のプログラムにも、こ
のAUTO命令をどんどん使ってください。

AUTO 20.....20から10間隔で

AUTO ,5..... 0から5間隔で

AUTO10から10間隔で



●RENUM(リナンバー)

プログラムを修正したり新しい命令文をつけ加えたりしているうちに、行番号が不ぞろいになったり、行と行がつまりすぎて新しい命令文をつけ加えることができなくなることがあります。そんなときに行番号を、好きなようにつけ直せる命令があれば便利です。それをするのがRENUM命令です。さっきのプログラムを使ってちょっと確かめてみましょう。

```
RENUM 1, 100, 1 RETURN
LIST RETURN
```

と打ってみましょう。行番号が1から1おきに変わっていますね。

```
RENUM 1, 100, 1
Ok
LIST
1 A=10
2 B=A*10+3
3 PRINT B
4 END
Ok
■
```

RENUM 1, 100, 1というのは「行番号100を1に変えて、そこから1おきに番号をつけ直しなさい」という意味の命令なのです。

```
RENUM 新行番号, 旧行番号, 間隔
```

RENUM とだけ書いてあとを省略すると、始めが1で以下10ずつ増えていく行番号につけ変わります。

```
RENUM RETURN
LIST RETURN
```

```
RENUM
Ok
LIST
10 A=10
20 B=A*10+3
30 PRINT B
40 END
Ok
■
```

ちゃんと変わっていますね。

```
RENUM 100 ..... 先頭(せんとう)の行番号(きょうばんごう)を 100
                           にして、10間隔(かんかく)にか
                           える
RENUM 100, 50... 行番号(きょうばんごう)50を 100 にし
                           て、それ以後(そのいご)を10間
                           隔(かく)に
RENUM , 50 ..... 行番号(きょうばんごう)50を10にして
                           それ以後(そのいご)を10間隔(かんかく)に
RENUM , 50, 5... 行番号(きょうばんごう)50を10にして
                           それ以後(そのいご)を5間隔(かんかく)に
```

●DELETE(デリート)

1行全部をプログラムから消してしまう方法は、行番号だけ打ち込んで **RETURN** とすればよいのでしたね。でも、多くの行をいっぺんに消したいときはどうすればよいのでしょうか。いちいち行番号を打ち込んで **RETURN** としていたのではたいへんですね。そんなときに便利なのが **DELETE** 命令です。

```
DELETE 消したい最初の行番号ー消したい最後の行番号
```

こうすることで消そうとする行が、プログラムから一度に消えることになります。さっきのプログラムで、ちょっとためてみましょう。まず **LIST** をとってみてください。行番号10から10おきになっていますね。そこで

```
DELETE 10-20 RETURN
LIST RETURN
```

としてみて下さい。行番号10、行番号20がなくなっているのがわかりますね。

```
DELETE 10-20
Ok
LIST
30 PRINT B
40 END
Ok
■
```

```
DELETE 150…行番号150を消す
DELETE -150…はじめてから行番号
150までを消す
```

●REM(リマーク)

REM はプログラムに説明を入れるための命令です。次のようなプログラムを作って **RUN** させてください。

```
10 REM テスト プログラム テス
20 REM A ニ 10ヲ イレマス
30 A=10
40 REM Aノ ナイヨウヲ ガメンニ
カキマス
50 PRINT A
60 REM オワリデス
70 END
```

このプログラムでそれぞれの行の **REM** 命令より後に書いたものは、それが何であっても実行されません。**REM** の後には好きなことを書いてよいのです。普通はプログラムをわかりやすくするために、プログラムの説明などを書いておきます。

```
10 REM テスト プログラム テス
20 REM A ニ 10ヲ イレマス
30 A=10
40 REM Aノ ナイヨウヲ ガメンニ カキマス
50 PRINT A
60 REM オワリデス
70 END
RUN
10
Ok
■
```

●REMの省略形

REM と書くかわりに、省略形「**'**」(アポストロフィ)記号を用いて、**REM** と同じ働きをさせることができます。たとえば上のプログラムの10行目、20行目はつぎのようになります。

```
10 ' テスト プログラム テス
20 ' A ニ 10 ヲ イレマス
```

GOTO文で流れをかえよう

…GOTOの使い方

(GOTO,CONT)

●GOTO (ゴートウー)

プログラムは行番号順に順序正しく実行していくと言いましたね。しかし、場合によっては順序をかえて実行させたいときがあります。こんなときに GOTO 命令が便利な働きをしてくれます。

次のようなプログラムを打ち込んでください。

```
10 A = 1
20 PRINT A
30 A = A + 1
40 GOTO 20
50 END
```

GOTO は、「そのあとに続く数字の行番号の所に飛びなさい」という意味です。ですから GOTO 20 は「行番号20へ飛びなさい」という意味になりますね。

GOTO 行番号

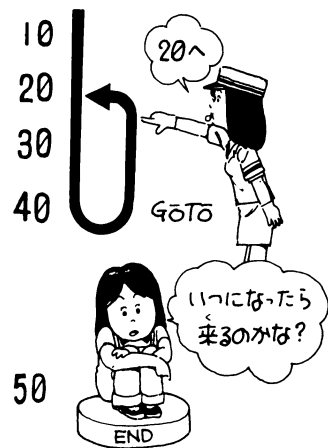
0 ~ 65529の数値
変数や式ではエラーになる

では、プログラムを実行してみましょう。

```
RUN
1
2
3
4
```

実際にはもっと数字が続いて表示されます。

1つずつ増えていく数が次々に画面に書かれていき、いつまでたっても止まりませんね。考えてみれば当然なのです。ベーシックは行番号40まで次々と実行していきませんが、40行の GOTO 20に出会って20行にとんでしまい、50行のENDは決して実行しないからです。



さて、それではどうすれば止まるでしょうか?

キーボードの左の方にある **CTRL** キーを押しながら上の方にある **STOP** キーを押してください。画面に

Break in 20 (30, 40のときもあります)

というメッセージが表示されて、プログラムは止まります。このメッセージは「プログラムの実行は、行番号20で止まりました」という意味です。

●CONT (コンティニュー)

ベーシックには、止めた所からもう一度実行を開始させる CONT 命令があります。

CONT RETURN

と打ち込んでください。STOP キーを押して止まった行番号の所から再び実行を始めるはずですが、実行を再開したくなければ、CONT 命令を実行する必要はありません。

```
101
102
103
Break in 20
Ok
CONT ←
104
```

実行を再開する。

(実際にはもっと数字が続いて表示されます)

〈起こりやすいエラー〉

●Undefined line number

(アンデファインド ライン ナンバー)

GOTO 文の飛び先である行番号がどこにもない場合に起こります。たとえば

GOTO 1000

とプログラムの中に書いてあるのに行番号1000がないような場合です。

●Can't CONTINUE

(キャント コンティニュー)

CONT 命令で、実行が再開できない場合に起こります。一度も RUN させていないのに、CONT 命令を実行させた場合などです。

「,」と「;」の大研究

…さらにくわしいPRINT命令

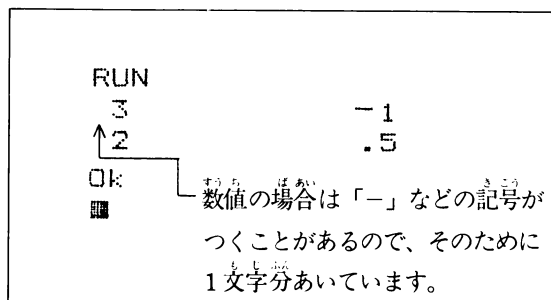
PRINT 命令は、プログラムの中でよく使われる命令です。これまで何度か出ています。ここでは、PRINT 命令といっしょに使うととても便利な「,」(カンマ)と「;」(セミコロン)について勉強しましょう。

```
10 A = 1
20 B = 2
30 C = A + B
40 D = A - B
50 E = A * B
60 F = A / B
70 PRINT C, D, E, F
80 END
```

とプログラムを打ち込んでください。行番号70に注目してください。

```
70 PRINT C, D, E, F
```

C, D, E, Fの間に「,」が入っていますね。この「,」はどんな働きをするのでしょうか?とにかくRUNさせてみましょう。

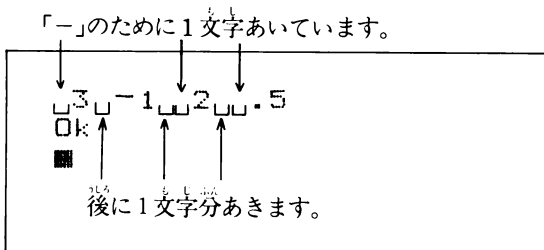


画面は図のようになります。PRINT 命令に「,」を使って変数をならべると、最初の変数の内容は始めの14文字分に、2つめの変数の内容はあとの14文字分にそれぞれ表示されます。(画面は29文字表示なので終わりの1文字分は残り、次の変数は次の行に表示されます。)次の変数も、その次の行に同じように表示されます。

今度は行番号70を次のように修正してください。

```
70 PRINT C; D; E; F
```

「,」を「;」に変えるだけです。修正の方法は覚えてありますね。では、さっそくRUNしてみましょう。



「;」は表示する文字や記号をぴったりとくっつけてしまうのです。変数だけでなく「"」「'」ではさんだ文字などの場合も同じことができます。次のように打ち込んでみましょう。

```
PRINT "ABC", "DEF" RETURN
PRINT "ABC"; "DEF" RETURN
```

ただし、「'」の場合には、長い文字列や小数点が入って14文字以上になると次の行に表示されます。

「;」では、文字はぴったりくっつきませんが、数字は前に「-」符号のために一文字分空白があき、後にも一文字分の空白があくのでぴったりくっつくことはありません。数字を文字として扱えば、くっつけることができます。次のように打ち込んで確かめてみましょう。

```
PRINT 10; 5; 3 RETURN
PRINT "10"; "5"; "3" RETURN
```

```
PRINT "ABC", "DEF"
ABC          DEF
Ok
PRINT "ABC"; "DEF" 文字の場合「-」
ABCDEF        のための空白と
Ok            後の空白はでない。
■
```

```
PRINT 10;5;3
10  5  3
Ok
PRINT "10";"5";"3"
1053
Ok
■
```

```
PRINT ○, ○, ○, .....
PRINT ○; ○; ○; .....
```



PRINTを助ける三銃士

…PRINTといっしょに使う便利な命令

や関数

(SPC関数, TAB関数, LOCATE)

PRINT 命令に「,」や「;」を組み合わせて使うと、画面にいろいろな書き方で文字が表示できましたね。H1にはPRINTと組み合わせて使う、もっと便利な命令や関数がいくつかあります。

●SPC(スペース) 関数

SPC関数は、好きな数だけスペース(空白)を画面に書くことができます。

SPC(あけたい空白の数)

0～255の数値、数値変数、式
かっこを忘れずに

```
PRINT SPC(5); "A"; SPC(6); 4*5
```

と打ち込んでみましょう。SPC(5)はスペース5つぶん、SPC(6)はスペース6つぶんということがよくわかりますね。

```
PRINT SPC(5); "A"; SPC(6); 4*5
```



```
Ok
PRINT SPC(5); "A"; SPC(6); 4*5
```

← A ← 20

Ok



6文字の空白

5文字の空白

「-」記号のための空白

PRINT 命令と後で紹介しますLPRINT命令の中でしか使えないので注意してください。

●TAB(タブ)関数

行の左はしから好きな数だけあけて書きはじめたいときには、TAB 関数を使います。

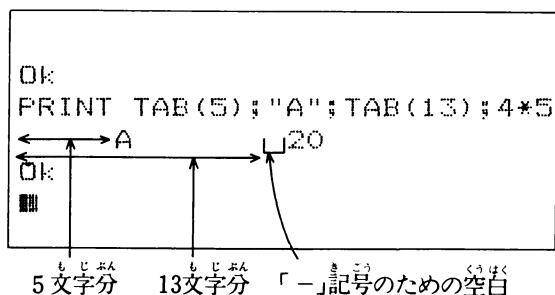
TAB (左からあけたい文字の数)

TAB 関数も PRINT 命令と LPRINT 命令の中でしか使うことができません。

今度は次のように打ち込んで下さい。

```
PRINT TAB(5); "A"; TAB(13); 4*5
```

RETURN

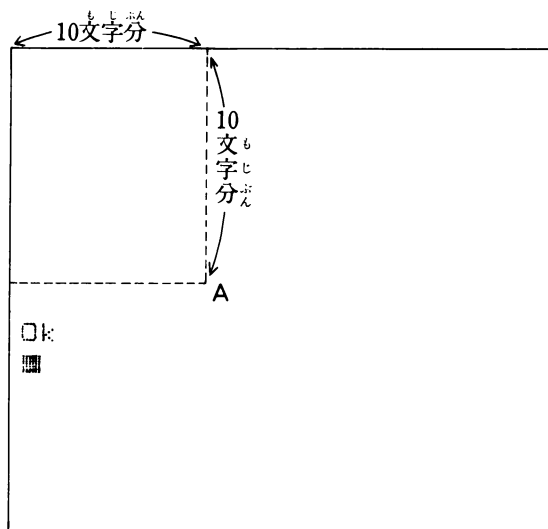


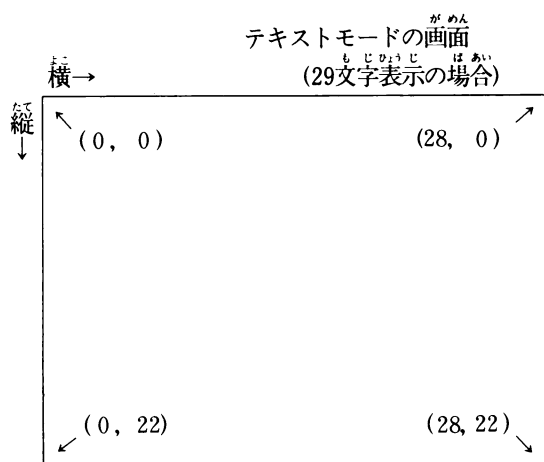
行の左はしを1個めと数えるので、TAB(5)は5文字分空白をあけるのと同じになるわけです。

●LOCATE(ロケイト)

画面にある小さな四角「■」をカーソルということはもう説明しましたね。PRINT 命令は、現在のカーソルがある場所に文字を書く命令です。ですから、プログラムで自由にカーソルを動かすことができれば、好きな場所から文字を書き始めることができるわけです。カーソルを動かす命令を LOCATE 命令といいます。

```
10 CLS
20 LOCATE 10, 10
30 PRINT "A"
40 END
```





〈起こりやすいエラー〉

● Illegal function call

(イリーガル ファンクション コール)

SPCやTAB, LOCATE に使う数値が決められた範囲をこえた場合に起こります。



くり返しに便利な命令

…FOR～NEXTの使い方

プログラムの中で、同じことを何度もくり返したい場合がよくあります。これまでは GOTO 文を使ってくり返しをして、**STOP** キーで終わらせていましたが、一定の回数だけくり返したら自動的に終わらせるというようにときに便利なのが、FOR～NEXT 命令です。

●FOR～NEXT

(フォー～ネクスト)

まず、1から10までの数を画面に書くプログラムを作って RUN させてみましょう。

```
RUN
1
2
3
4
5
6
7
8
9
10
Ok
■
```

```
10 FOR N = 1 TO 10
20 PRINT N
30 NEXT N
40 END
```

```
10 Nを1から10まで1ずつふやしながら、次のことをくり返しなさい。
20 Nのなかみを画面に書きなさい。
30 ここまでをくり返しなさい。
40 終わりです。
```



このプログラムは
こういう意味なのね

FOR～NEXT命令は、FOR～とNEXT～の間に囲まれた部分（このプログラムでは、20 PRINT Nがその部分です）をある決められた回数だけくり返す命令です。くり返しの数を覚えている変数（制御変数ともいいます）を使えるので、とても便利です。

FOR 変数=初めの値 TO 終りの値

↑ ↑ ↑ ↑

変数、式でもよい。

「=」イコールです。

くり返しの数を覚えておくための変数

くり返す命令(いくつ書いてもよい)

NEXT 変数

↑

FORの次に書いた変数と同じものを使うこと。

この変数の変化を1, 2, 3と1ずつ大きくするのではなく、たとえば0.5ずつ大きくするとか、あるいは2ずつ小さくすることもできます。このときの変化幅をSTEPという命令で指定します。

FOR I = 1 TO 10 STEP 0.5 ←

FOR A = 10 TO -10 STEP -2 ←

変数、式でよい。

それでは、さきほどのプログラムを、2, 4, 6, ... 10と書くプログラムに修正してみましょう。

10 FOR N = 2 TO 10 STEP 2 RETURN

四角で囲んだところだけを修正するのですよ。できたら、RUN させてみましょう。

```
RUN
2
4
6
8
10
OK

```

●もっと複雑なFOR～NEXTにチャレンジ

FOR～TO～NEXT 命令は、2重3重に重ねて使うことができます。しかし、必ず守らなければならない規則があります。

- (1) 内側のFOR～NEXT と外側のFOR～NEXT 命令が交差してはいけません。
- (2) FOR～NEXT 命令の内側からGOTO 命令やIF～THEN 命令(122 ページ)などで外側に飛び出すことはできますが、FOR～NEXT 命令の外側から内側に飛び込んではいけません。

```

FOR A = 1 TO 10
  FOR B = 2 TO 5
    FOR C = 3 TO 4
      <繰り返す命令>
    NEXT C
  NEXT B
NEXT A

```

●よい例

```

FOR A = 1 TO 10
  FOR B = 2 TO 5
    FOR C = 3 TO 4
      <繰り返す命令>
    NEXT A
  NEXT B
NEXT C

```

●悪い例

FOR～NEXT は交差してはいけません。

```

50 GOTO 200
120 FOR I = 0 TO N
200 .....
270 NEXT I

```

●悪い例

FOR～NEXT への飛び込みはいけません。

それでは、2重のFOR～NEXT 命令を使って、九九の計算表を作るプログラムを書いてみましょう。

```
10 CLS
20 FOR X = 1 TO 9
30 FOR Y = 1 TO 9
40 LOCATE 3 * X - 3, 2 * Y :
   PRINT X * Y
50 NEXT Y
60 NEXT X
70 END
```

変数XのFOR～NEXT 命令のくり返しの中に、変数YのFOR～NEXT 命令のくり返しが含まれています。変数Xと変数Yはそれぞれ1から9まで変化しますが、最初に変数Xに1がはいり、Y = 1, 2, …, …, 9とくり返します。変数Yが9までいくと、内側のくり返しは終わりになり、今度はX = 2となり同じことをくり返すわけです。

では、さっそくRUNさせてみましょう。

```
RUN
 1  2  3  4  5  6  7  8  9
 2  4  6  8 10 12 14 16 18
 3  6  9 12 15 18 21 24 27
 4  8 12 16 20 24 28 32 36
 5 10 15 20 25 30 35 40 45
 6 12 18 24 30 36 42 48 54
 7 14 21 28 35 42 49 56 63
 8 16 24 32 40 48 56 64 72
 9 18 27 36 45 54 63 72 81
Ok
■
```

きれいな九九の表が出来上りました。

〈よく起こるエラー〉

●NEXT without FOR

(ネクスト ウィズアウト フォー)

FORを実行していないのにNEXTを実行したときに起こります。FOR～NEXT 命令の外側から内側に飛び込んだときや、FORの制御変数とNEXTの制御変数が違っている場合などに起こります。

も　じ　れつ　なに 文字列って何？

も　じ　れつ　つか　かた …文字列の使い方

●文字変数を使おう

文字列というのは数字や、アルファベット、記号などを PRINT 命令の中などで「”」（ダブルクォート）でくくったものです。たとえば

”ABCDEFG” や ”わたしは MB-H1 です”

などは文字列です。「”」の中に使える文字は、H1 から打てる文字なら、どのようなものを使ってもかまいません。ただし、「”」だけは使うことができません。では文字列を画面に書いてみましょう。

PRINT ”わたしは MB-H1 です” RETURN

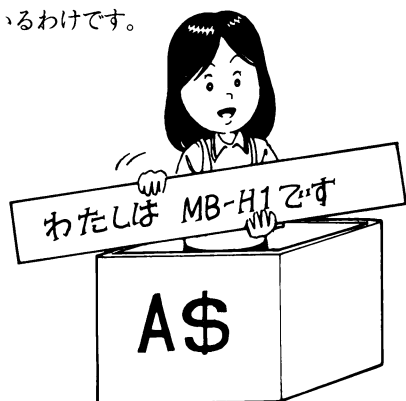
とすればよいのでしたね。

文字列を使うことによってコンピュータが人間にわかる言葉を作り出すのです。

いままで使ってきた変数は数を表すものでした。変数には、数を表わすものばかりでなく、文字列を表すものもあります。それを文字変数といいます。次のように打ち込んでください。

A\$ = ”わたしは MB-H1 です” RETURN
PRINT A\$ RETURN

PRINT ”わたしは MB-H1 です”としたときと同じ結果になりましたね。ここで使った「A\$」が文字変数で、「わたしは MB-H1 です」という文字列がはいっているわけです。



これが文字変数か



文字変数の名前の最後に「\$」（ドル）記号をつけます。この記号をつけることで、これまでにできた変数（数値変数といいます）と区別しているわけです。名前のつけ方は 81 ページの「変数名の 3 つの規則」をよく読んでください。

●文字列のつなぎあわせ

文字列や文字変数は数ではないので、たし算、引き算、かけ算、わり算などの四則演算はできません。ですからPRINT "123" + 25やPRINT "2" * 3 + 2などという計算を行なうとエラーになります。

そのかわり、つないだり、切ったり、抜きだしたりすることができます。ここでは、文字列のつなぎ方について説明しましょう。

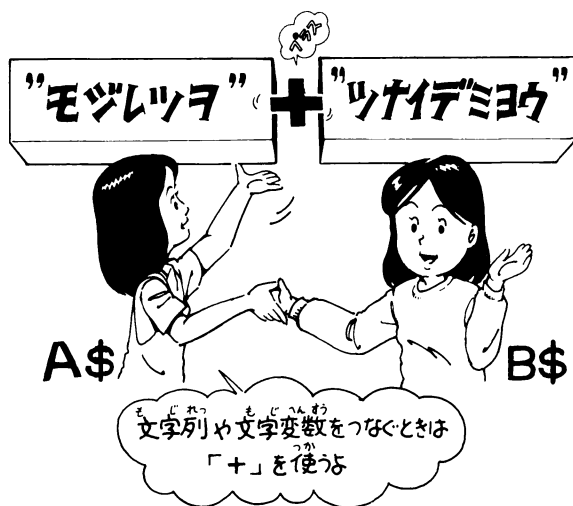
```
PRINT "モジレツ ヲ "+"ツナイデ  
ミヨウ"
```

と打ち込んでください。「モジレツ ヲ ツナイデ ミヨウ」と1つにつながって表示されますね。今度は

```
10 A$="モジレツ ヲ "  
20 B$="ツナイデ ミヨウ"  
30 C$=A$+B$  
40 PRINT C$  
50 END
```

としてRUNさせてください。同じ結果になりましたね。

このようにいくつかの文字列や文字変数を「+」によってつなぐことができるのです。これを文字式といいます。ここで使った「+」はたし算の「+」とは意味が違うことに注意してください。



〈起こりやすいエラー〉

●Type mismatch (タイプ ミスマッチ)

数値と文字列を混同して使った場合などに起こります。

たとえば

```
PRINT "123" + 123
```

"123"は文字列ですが1 2 3は数値です。また

```
PRINT A$ + B
```

A\$は文字変数ですがBは数値変数です。いずれの場合もエラーになります。

文字列料理の3大道具

…文字列を抜き出そう

LEFT\$関数, RIGHT\$関数, MID\$関数, LEN関数

文字列のつながりかはわかりましたね。ここでは文字列を切ったり、抜き出したりするときに使う便利な道具について説明しましょう。

●LEFT\$ (レフトドル) 関数

文字列や、文字変数の左から何文字かを抜き出すときに使います。

かっことも忘れないこと。

文字の式でもよい。

「,」(カンマ)を忘れずに

数値変数や式でもよい。

LEFT\$ (文字列か文字変数, 抜き出す文字数)

次のプログラムを打ち込んでください。

```
10 FOR I = 1 TO 10
20 PRINT LEFT$ ("1 2 3 4 5 6 7 8
9 0", I)
30 NEXT I
40 END
```

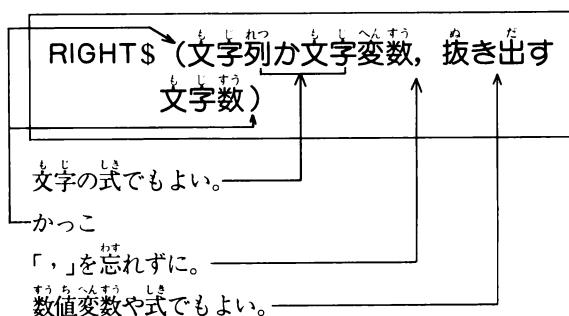
最初は左から1文字 (つまり「1」)、次に2文字 (「12」)、3文字…と、10文字まで「1 2 3 4 5 6 7 8 9 0」を左から抜き出していくプログラムです。さっそくRUNさせてみましょう。

```
RUN
1
12
123
1234
12345
123456
1234567
12345678
123456789
1234567890
OK
■
```

うまくいきましたね。

●RIGHT\$(ライトドル)関数^{かんすう}

文字列や、文字変数の右から何文字かを抜き出すときに使います。



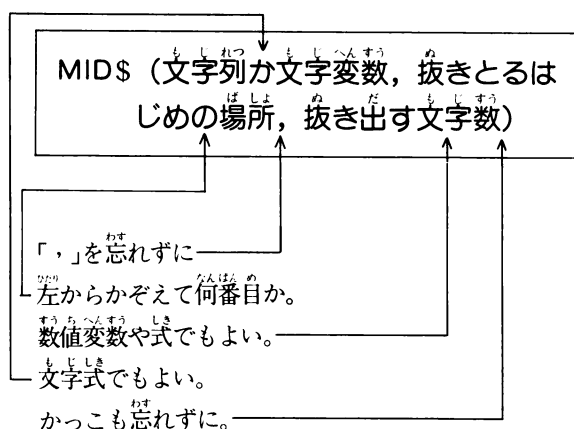
同じように次のプログラムを RUN させてみましょう。

```
10 FOR I = 1 TO 10
20 PRINT RIGHT$("1 2 3 4 5 6 7 8
  9 0", I)
30 NEXT I
40 END
```

```
RUN
0
90
890
7890
67890
567890
4567890
34567890
234567890
1234567890
Ok
■
```

●MID\$(ミドルドル)関数^{かんすう}

文字列や文字変数の、好きな場所から好きな文字数だけ抜き出すときに使います。



```
10 FOR I = 1 TO 10
20 PRINT MID$("1 2 3 4 5 6 7 8
  9 0", I, 11-I)
30 NEXT I
40 END
```

```
RUN
1234567890
234567890
34567890
4567890
567890
67890
7890
890
90
0
Ok
■
```

どうですか？ 左と右が反対になっただけで同じようなことが起きているのがわかりますね。

●LEN (レングス) 関数

最後に今までのRIGHT\$, LEFT\$, MID\$ とはちよつと違った働きをもつ関数を紹介しましょう。

LEN関数は文字列の長さを与えるものです。

```
PRINT LEN("ABCDEF")
```

としてみましょう。

答えは6となりますね。

LEN (文字列)

それでは次のようにプログラムして、RUN させてみましょう。

```
10 A$="1 2 3 4 5 6 7 8 9 0"  
20 FOR I=1 TO 9  
30 PRINT MID$(A$,LEN(A$)  
- I, 2)  
40 NEXT I  
50 END
```

```
RUN  
90  
89  
78  
67  
56  
45  
34  
23  
12  
0k  
■
```

「1 2 3 4 5 6 7 8 9 0」を右から順番に2文字ずつ表示していくのがわかりますね。

〈起こりやすいエラー〉

●Type mismatch

(タイプ ミスマッチ)

文字と数値を使い間違えたときに起こります。たとえば

```
MID$(A, 2, 3)
```

など、文字変数を入れるべきところに、数値変数(この場合A)を入れた場合です。

こちらINPUT応答せよ

コンピュータに^{へんすう}変数の^{あたい}値を知らせるために、いままでは、

```
10  A = 2
20  B = 5
   :
```

のように、^{はじめ}最初に^{すうち}数値を決めてしまう^{ほうほう}方法をとってきました。^{へんすう}変数の^{あたい}値を変えたい^{ばあい}場合には、プログラムを^{しゆせい}修正しなければならないのです。そこで、^{ひつよう}必要に応じて^{おう}キーボードから^{へんすう}変数の^{あたい}値を変えられる^{めいれい}INPUT 命令の^{とうじやう}登場です。

●INPUT (インプット)

^{えん}円の^{めんせき}面積を求める^{しと}次の^{つぎ}プログラムを^う打ち込んで^うRUN
させてみましょう。

```
10  INPUT  R
20  S=3.14159*R*R
30  PRINT  S
40  GOTO  10
50  END
```

^み見られない^{ひょうじ}表示が^で出てきましたね。「?」(クエスチョン
マーク) は、コンピュータがあなたに、「^{へんすう}変数Rの^{あたい}値は
^{なん}何にしますか?」ときいてきているのです。

たとえば、

10 RETURN

と^う打ち込んでみましょう。コンピュータは、 $3.14159 \times 10 \times 10$ を^{けいさん}計算して、

314.159

と^{こた}答えを出す^だはずです。いろいろな^{かず}数を入力^{にゅうりく}してため
してみてください。

RUN
? █ ←カーソル

```
RUN
? 10
  314.159
? 1
  3.14159
? 2.2
 15.2052956
? ← ここで CTRL
Break in 10  + STOP キー
Ok           を押した
█
```



● たくさんの変数への入力

さきほどのプログラムでは、値を入力する変数はRただ1つでしたが、変数がたくさんある場合はどうしたらよいでしょうか。

次のプログラムは、一度に入力された4つの数の平均を出すプログラムです。

```
10 INPUT A, B, C, D
20 PRINT (A+B+C+D)/4
30 GOTO 10
40 END
```

```

RUN
? 10,20,30,40
  25
? 1,2,3,4
  2.5
? ←----- ここで [CTRL] + [STOP]
Break in 10      キーを押した。
Ok

```

RUNさせて「?」がでてきたら、数字を打ち込みます。
たとえば、

10, 20, 30, 40 [RETURN]

と入力してください。打ちまちがえたときは、[RETURN]を押す前ならば、カーソルキーや[DEL]、[INS]、[BS]キーで修正することができます。

たくさんの変数に INPUT 命令で値を入れる場合は
「・」をはさんで入れることを覚えておいてください。

●文字変数にも入力できるよ

今度は、

INPUT 変数, 変数, …… , 変数

```
10 INPUT A$, B$
20 PRINT B$; A$
30 GOTO 10
40 END
```

として RUN してみましょう。

前ページのプログラムを RUN させて、次のように数
値を入力してみましょう。

15, 20, 30 RETURN

入力する変数の値が1つ少ないですね。

H1は、「?」(疑問符)を2つ表示します。
これは入力される変数の個数が足りないという意味
ですから、つづけて、

35 RETURN

とします。

```
RUN
? AAA, BBB
BBBAAA
? ME-H1 テ"ス, ワクシハ
ワクシハME-H1 テ"ス
?
Break in 10
OK
■
```

INPUT 命令を使って文字変数
に値を入れるときには、「"」で
囲む必要はない。

ここで CTRL + STOP キー
を押した。

文字変数にも値(文字列)を入力できることがわかり
ますね。

2つ以上を一度に入力するときに「・」を間に入れるこ
とは、数値を入力する場合と同じです。

INPUTで書いた変数の個数が入力した値の
個数よりも多いときは「?」となります。
このときはつづけて入れます
反対に、入力した値の個数の方が多いときは、
「Extra ignored」となり、入力された値のうち、
INPUTで書いた変数の個数分だけ H1は読みとります



● INPUT 命令で画面表示

INPUT 命令を使った場合、画面に

？ ■ ← カーソル

と出てきますが、これでは、数値を要求しているのか、文字列を要求しているのか、あるいは入力する個数はいくつなのか、まったくわかりませんね。
そこで、INPUT 命令の後に、「”」でかこんだ文字列を置くと、その文字列を、「？」といっしょに表示してくれます。

INPUT "文字列"; 変数, 変数, …… , 変数

それでは、さっき作ったプログラムの行番号10を次のように書き直してRUNしてみましょう。

10 INPUT "モジヲ 2ツ"; A\$, B\$

```
RUN
モジヲ 2ツ? ABC, DEF
DEFABC
モジヲ 2ツ? MB-H1 テス, ワタシハ
ワタシハMB-H1テス
モジヲ 2ツ?
Break in 10
Ok
■
```

— [CTRL] + [STOP] キー

このほうが、わかりやすいですね。

お 起こりやすいエラー>

● Redo from start (リドゥー フロム スタート)

数値変数へ入力する INPUT 命令なのに、実際に入力されたものが、数値でなかった場合に起こります。
たとえば、

10 INPUT A

というプログラムで、数値を入力するべきなのに、

？ ABC [RETURN]

などと、文字を入力してしまった場合です。

● Extra ignored (エクストラ イグノアー ド)

INPUT で書かれた変数の個数よりも、キーで入力された値の個数が多いときに起ります。入力された値は、変数の個数の分だけ、先頭から読みこまれ、残りは無視されます。

10 INPUT A, B, C

というプログラムで、

？ 10, 20, 40, 50, 60 [RETURN]

とすると、50, 60は無視されます。

数の料理の専門家

… 数値関数

アブソリュート かん すう サイン かん すう インテジャー かん すう
(ABS関数,SGN関数,INT関数)

もじれつや文字変数を料理する道具 (LEFT\$,RIGHT\$, MID\$) については、もうお話ししましたね。ここでは、数値を料理する関数を説明しましょう。

●ABS(アブソリュート)関数

絶対値という言葉聞いたことがありますか？ 正の数はそのままに、負の数は「-」(マイナス)を取ったものを値とするのが絶対値です。たとえば次のように使います。

5の絶対値は5
-3の絶対値は3
0の絶対値は0

この絶対値を与えるのがABS関数です。

ABS(数値か変数か式)←

かっこを忘れずに

次のプログラムをRUNさせてみましょう。

```
10 INPUT X
20 PRINT ABS (X)
30 GOTO 10
```

正の数や、負の数をキーボードから入力してください。
「-」がなくなって画面に表示されますね。

```
RUN
? 10.5
10.5
? -10.5
10.5
? 0
0
? ←
Break in 10
Ok
■
```

CTRL + STOP キー

●SGN(サイン)関数

正の数ならば1、負の数ならば-1、0ならば0というように数の符号を与えます。

SGN(数値カ変数カ式)

かっこを忘れないこと

使い方は、ABS関数と同じです。さっきのプログラムの行番号20を次のように変えて、RUNしてみましょう。

20 PRINT SGN(X)

```
RUN
? -20
-1
? 10.23
1
? 0
0
? 
Break in 10
Ok
```

CTRL + STOP キー

●INT(インテジャー)関数

INT関数は、小数点以下の部分を切り捨てて、整数にする命令です。

INT(数値カ変数カ式)

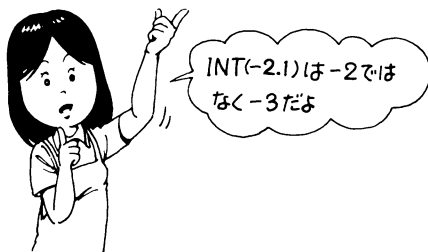
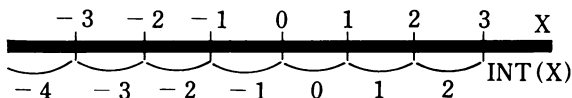
今度は、行番号20を次のようにして、RUNさせてみましょう。

20 PRINT INT(X)

```
RUN
? 1.5
1
? 0.01
0
? -2.1
-3
? -0.01
-1
? 0
0
? 
Break in 10
Ok
```

CTRL + STOP キー

INT関数では、与えられた数値の小数点以下の部分を、負の方向に切り捨てます。たとえば、4.25のときに4、-5.3のときには-6となりますから、INT関数では、その結果が、与えられた数値よりも、必ず小さくなります。



●ちよつと関数のお話

ABS、SGN、INTなどの数を料理する専門家を説明してきました。これらは関数と呼ばれています。関数とは「ある数値が与えられ、それになんらかの処理をした結果を値として持つもの」をいいます。たとえば、ABS(−2)というのは、−2という数が与えられて、それに絶対値をとるという処理をした結果の、2という値を持っています。SGN、INTも同じことが言えますね。

関数は、それ自身、値を持つので、式の中を書くこともできるし、PRINT命令の右側に書くこともできるのです。それで、

```
PRINT INT(5.6)*SGN(−6)+3
```

などをちゃんと計算してくれますね。

ところで、すでに説明したLEFT\$, RIGHT\$, MID\$なども似たような性質を持っていると思いませんか？

LEFT\$("ABCD", 2)というのは、"ABCD"という文字列と2という数が与えられて、「与えられた文字列の左から、与えられた数字の分だけ取り出す」という処理をした結果、"AB"という値を持っています。

また、

```
PRINT LEFT$ ("1234", 2) + "56"
```

などと文字式の中に使うことができます。RIGHT\$, MID\$も同じことが言えます。

LEFT\$, RIGHT\$, MID\$も関数なのです。ただし、値として持つのが文字列なので、これらは文字関数と呼ばれています。それに対して、ABS、SGN、INTなどは数値関数と呼ばれています。

●その他の関数

H1はこの他にも、たくさんの関数を持っています。詳しいことはベーシック文法編をお読みください。

でたらめな数をRND関数で作ろう

ここまでH1とつきあってきますと「コンピュータは規則正しいことはできるが、でたらめなことはできない」あなたはそう思っていないでしたか。ところが違うのです。H1にはでたらめな数を作り出すRND関数があります。

●RND(ランダム)関数

RND関数は、0より大きい1未満のでたらめな数(乱数といいます)を作る関数です。普通は、

RND(1)

と書きますが、「1」のところは正の数なら何でもかまいません。すべて同じ結果になります。

RND関数を使ってコンピュータ・サイコロを作ってみましょう。

サイコロのワク組の線はグラフィック文字ですから、

GRAPH キーを押しながら書いてください。

```
10 X=INT(RND(1)*6)+1
20 PRINT "┌───┐"
30 PRINT "│□□□│"
40 PRINT "│"; X; "│"
50 PRINT "│□□□│"
60 PRINT "└───┘"
```

```
70 INPUT "モウイチド "; A$
80 GOTO 10
90 END
```

行番号10の

INT (RND(1)*6)+1

は次のような計算をしているのです。

RND(1)	0より大きく1より小さい
RND(1)*6	0より大きく6より小さい
INT(RND(1)*6)	0,1,2,3,4,5のどれか
INT(RND(1)*6)+1	1,2,3,4,5,6のどれか

となるので、ここでサイコロの目を計算していることがわかります。サイコロの目のように、結果が不規則なものを作り出すにはRND関数を使うことが便利ということがわかりますね。

それではさっそくRUNしてみましょう。

モウイチド?

と表示されたら適当な文字や数値を入れて **RETURN** キーを押してください。

RUN

5

モウイチト？ Y

てきとう
適当なキーを押
して RETURN
としてください。

3

モウイチト？ Y

CTRL + STOP
キーを
押した


6


モウイチト？
Break in 70
Ok

りっぱなコンピュータ・サイコロのできあがりです。

「:」を使ってプログラムを短く …マルチステートメントの使い方

今までは、1つの行に1つの命令しか書きませんでした。ベーシックでは「:」(コロン)で区切ることによって、1行にいくつでも命令を書くことができます。たとえば次のように書くことができます。

```
10 A=5:B=10:C=20
20 PRINT A+B+C:PRINT A*B*C:PRINT A/B/C:END
RUN
35
1000
.025
Ok

```

```
10 A=5
20 B=10
30 C=20
40 PRINT A+B+C
50 PRINT A*B*C
60 PRINT A/B/C
70 END
RUN
35
1000
.025
Ok

```

書きかたは違うけれど、
実は同じプログラムです。

このように、「:」を使って1行にいくつもの命令を書いたものを、マルチステートメントと言います。長いプログラムを書くときなど、マルチステートメントにすると行番号を書かないで済むので、プログラムが短くなり、実行速度が多少速くなるなどの利点があります。しかし、1行にあまりたくさん詰め込むと、プログラムが見にくくなるので、あまりマルチステートメントを多用するのはやめましょう。

判断はIF～THENにおまかせ

(IF～THEN～ELSE)

H1をますますコンピュータらしく使う命令を紹介しましょう。条件判断をする IF～THEN 命令です。

●IF～THEN(イフ～ゼン)

IF～THEN 命令は「もし～だったら…」という条件判断を行なう場合に使います。

```
30 IF△△△THEN□□□  
40 ■■■■
```

「もし△△△の条件が満たされるのならば、□□□ を実行すること。満たされないならば□□□は無視して次の行の■■■■を実行しなさい」というのが IF～THEN 命令です。

次のプログラムを RUN させてみましょう。

```
10 INPUT X  
20 IF X>10 THEN PRINT  
   "10ヨリオオキイ":GOTO 10  
30 PRINT "10 イカデス"  
40 GOTO 10  
50 END
```

「？」が出てカーソルが出ますので、何か数字を入れて RETURN としてください。入れた数が10より大きいときは、

10ヨリオオキイ

10以下のときは、

10 イカデス

と答えてくれますね。


```

RUN
? 2
10 イカデ`ス
? 46
10ヨリオオキイ
? 777
10ヨリオオキイ
? -98
10 イカデ`ス
? 4.873
10 イカデ`ス
? 0.987
10 イカデ`ス
? -100.45
10 イカデ`ス
? 46764878
10ヨリオオキイ
? ←

```

```

Break in 10
Ok

```

[CTRL] + [STOP] キー

プログラムの行番号20にある $X > 10$ のように2つの値を比較する式を条件式と呼びます。

IF 条件式 THEN 命令 : 命令 : ……

マルチステートメントでつなげれば、THENのあとに命令を続けて書くことができる。

条件が満たされた場合、ここが実行される。

条件式の書きかた

記号	意味	書きかた
=	AとBは等しい	A = B
>	AはBより大きい	A > B
<	AはBより小さい	A < B
<>または><	AとBは等しくない	A <> B
>=または=>	AはB以上	A >= B
<=または=<	AはB以下	A <= B

また、THENのすぐ後にGOTO命令を書くこともできます。

IF 条件式 THEN GOTO 行番号

このときは、GOTO命令か、THEN のどちらかをはぶくことができます。

IF 条件式 THEN 行番号

IF 条件式 GOTO 行番号

と、どちらでもかまいません。

●ELSE(エルス)

IF~THEN 命令に ELSE をつけ加えて IF~THEN ~ELSE という命令にすることができます。

IF△△△THEN□□□ELSE■■■■

「もし△△△の条件が満たされるのならば□□□を実行し、満たされないのならば□□□を飛ばして■■■■を実行しなさい」という命令になります。条件式や、命令の書き方は IF~THEN 命令と同じです。

さきほどのプログラムを IF~THEN~ELSE を使つて書き換えてみましょう。

```

10 INPUT X
20 IF X>10 THEN PRINT "10ヨリオオキイ"
   ELSE PRINT "10イカデ`ス"
30 GOTO 10
40 END

```

この方がプログラムは簡単になりますね。RUN の結果は、さっきと同じですね。

また、

IF 条件式 THEN 命令 ELSE
GOTO 行番号

のときは、GOTO命令をはぶいて、

IF 条件式 THEN 命令 ELSE 行番号

と書くことができます。

●AND, OR (アンド, オア)

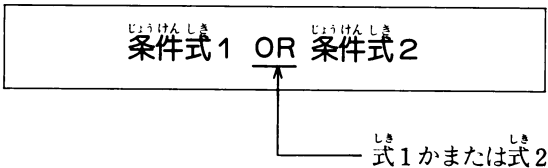
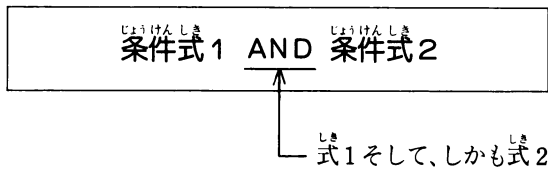
複雑な条件式を使う時に、このAND, ORが役に立ちます。たとえば、「Xが10以上であって、しかも20以下」という条件式は、ANDを使って、

$X \geq 10 \text{ AND } X \leq 20$

と簡単に書くことができます。また、「Xが負の数か、または100以上」という条件式はORを使って、

$X < 0 \text{ OR } X \geq 100$

とすれば簡単に書くことができます。



身長と体重の差が100より大きく、110より小さいなら「ベストコンディション」、そうでないなら、「フトリスギ マタハ ヤセスギデス!」と出力するプログラムを作ってみましょう。

```

10 INPUT "シンチョウcm, タイ
    ジュウkg"; X, Y
20 Z=X-Y
30 IF Z>100 AND Z<110
    THEN PRINT "ベストコン
        ディション" ELSE PRINT
        "フトリスギ マタハ ヤセス
        ギデス"
40 GOTO 10
50 END
    
```

```

RUN
シンチョウcm, タイジュウkg? 173,60
フトリスギ マタハ ヤセスギデス
シンチョウcm, タイジュウkg? 173,65
ベストコンディション
シンチョウcm, タイジュウkg? 153,48
ベストコンディション
シンチョウcm, タイジュウkg? ←
Break in 10
Ok
■
    
```

CTRL + STOP キー

ゲームをするなら INKEY\$ 関数が便利

前に勉強した INPUT 命令は、最後に **RETURN** としないと入力を受けつけてくれませんでしたね。また、押したキーの文字を画面に書いてしまうので、インベーダーゲームのような、画面を使うゲームには使うことができません。そこで登場するのが INKEY\$ 関数です。

何か、キーを押してみてください。キーを押していれば画面に、押したキーの文字を表示します。キーを押していなければ、画面に何も書きませんね。つまり、INKEY\$ 関数というのは、そのとき押されていたキーの文字を与える命令なのです。

● INKEY\$(インキードル) 関数

まず、次のプログラムを RUN させてみましょう。

```
10 A$=INKEY$
20 PRINT A$:GOTO 10
30 END
```

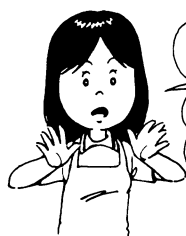
```
RUN
A
A
A

!

1
1
Break in 10 (20の場合もあります。)
```

INKEY\$

CTRL +
STOP キー



文字を与えることに注意してね
A=INKEY\$では
Type mismatch エラーだよ。
A\$=INKEY\$としてね

INKEY\$関数は、INPUT 命令のように、キー入力があるまで実行を中断してくれません。キーが押されていなくても、さっさと次の命令に移ってしまいますから、このプログラムのように GOTO 命令などを使ってくり返し INKEY\$ 関数を実行させてください。

また、INKEY\$ 関数は、キーを押してもその文字を画面には表示しないので注意してください。画面に表示させたい場合は、このプログラムのように PRINT 命令を実行させなければなりません。

●INKEY\$ をゲームに使おう

ちょっとゲームの基礎の基礎を勉強してみましょう。押したキーによって前後左右に絵を動かすプログラムです。

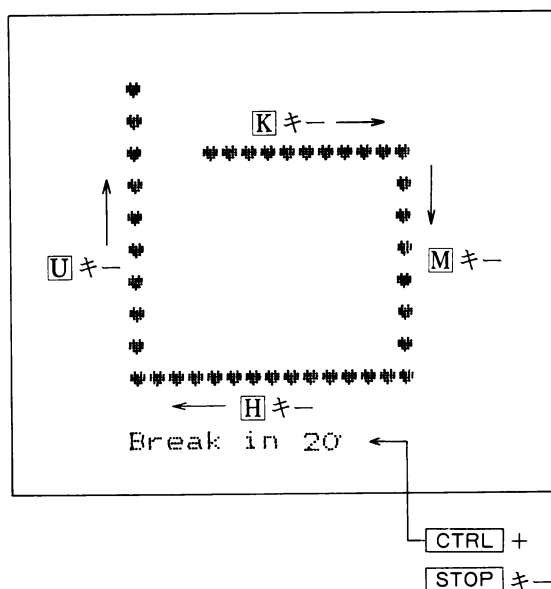
```

10 X=10:Y=10:CLS
20 A$=INKEY$
30 IF A$="U" THEN
    Y=Y-1:GOTO 70
40 IF A$="H" THEN
    X=X-1:GOTO 70
50 IF A$="M" THEN
    Y=Y+1:GOTO 70
60 IF A$="K" THEN
    X=X+1
70 LOCATE X,Y:PRINT
    "♥"
80 GOTO 20
90 END

```

(「♥」はグラフィック文字です)

さあ、RUN させてみましょう。Uを押すと「♥」マークが上に、Hを押すと左に、Mだと下に、Kを押すと右に動きますね。



ベーシックで作られたゲームはたくさんありますが、「キーを押すと絵が動く」という部分は、基本的にはこのプログラムと同じものです。COLOR 命令を使ったり、もっと複雑な絵を描かせたりすれば、楽しいゲームが出来上がります。

〈起こりやすいエラー〉

●Type mismatch(タイプ ミスマッチ)

数値変数と文字変数の使い間違いなどで起こります。

デバッグのお話^{はなし}

ベーシックにはだいふなれましたか？ 命令を理解するとともにエラーのおこる回数も増えてきますね。エラーの起こる原因をバグ（虫という意味です）と呼びます。ここでは、バグをなくす方法 デバッグ（「虫取り」という意味になります）、について説明します。まず次のプログラムを打ち込んでみましょう。これは、簡単なルーレット・ゲームのプログラムです。

```

10 DIM C$(2),X$(2)
20 C$(0)="♠":C$(1)="♥":C$(2)="♦"
30 K=1000
40 CLS
50 IF K<=0 THEN 300
60 LOCATE 7,2
70 PRINT "モチテン_:_";K
80 LOCATE 7,4
90 INPUT "カケテン_:_":L
100 IF L>K THEN BEEP:GOTO 60
110 FOR I=0 TO 2
120 X$(I)=C$(RND(1)*3)
130 NEXT I
140 LOCATE 11,10
150 FOR I=0 TO 2:PRINT X$(I);:NEXT I
160 IF INKEY$=" " THEN 110
170 LOCATE 6,16
180 PRINT"_____ "
190 LOCATE 6,16
200 P=1
210 IF X$(0)=X$(1) THEN P=P+1
220 IF X$(1)=X$(2) THEN P=P+1
230 IF X$(0)=X$(2) THEN P=P+1
240 IF P=1 THEN 270
250 IF P=2 THEN 280
260 IF P>=3 THEN 290
270 PRINT "ハズレ_マイナス_";4*L:K=K-4*L:GOTO 50
280 PRINT "アタリ!プラス_";L:K=K+L:GOTO 50
290 PRINT "オオアタリ!!プラス";5*L:K=K+5*L:GOTO 50
300 FOR I=1 TO 10:BEEP:NEXT I
310 CLS:LOCATE 9,10
320 PRINT "ハサンデス!!":END

```

注) 「_」は、^{ひとじぶん}一文字分の^{くうはく}空白(スペース)を^{いみ}意味します。

注) 「 」は、^{ひとし　じぶん　くうはく}一文字分の空白(スペース)を意味しま
す。

●ルーレット・ゲーム

今までのプログラムよりちょっと長くなっています。
うまく打ち込めましたか？ 打ち込んだらさっそく
RUN してみましょう。

モチテン：1000

カケテン：？ ■

と画面に出てきましたね。「あなたは今1000点持っています。何点かけますか？」とコンピュータが聞いているのです。持ち点以内の数字を打ち込んで「RETURN」としてください。「♠」、「♥」、「♦」の記号が、画面の中央に、めまぐるしく移り変わりながら表示されますね。適当なところでスペースキーを押すと表示がそこでストップします。そこで同じ記号が2つ出れば「アタリ!」、3つとも同じならば「オオアタリ!!」、3つとも違っていれば、「ハズレ」と表示して、持ち点が増えたり減ったりするわけです。

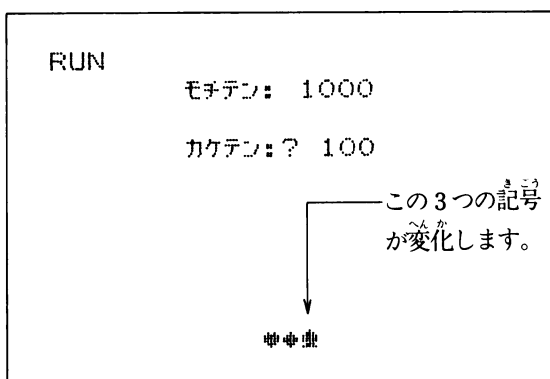
「アタリ!」の場合、かけ点ぶんの点数が持ち点に加えられます。

「オオアタリ!!」の場合は、なんとかけ点の5倍の点を増やしてくれます。ただし、「ハズレ」の場合、かけ点の4倍の点が、持ち点からさしひかれますよ。

持ち点が0点以下になると、ブザーが鳴り、「ハサundes!!」と表示してゲームは終りになります。

さあ、あなたはコンピュータ・ルーレットで何点もうけることができますか？ さあ、ルーレットを楽しんでみましょう。

コンピュータ・ルーレットはうまく動きましたか？ プログラムが長くなってくるとエラーも多くなってきました。ここで、デバッグの方法をすこし勉強しましょう。コンピュータ・ルーレットがうまく動いた人も、必ず読んでください。



●間違いがどこかわかっている場合

たとえば

シンタックス エラー イン
Syntax error in 100

と出たとします。これは行番号100に、書き間違いがあるという意味ですね。そのときは、

LIST 100 RETURN

として、画面にエラーの起こった行を表示し、カーソルコントロールキーを使って、間違った文字をなおしていくのでしたね。実はもっと簡単な方法があります。

LIST. RETURN

と、LISTの後に「.」(ピリオド)を書くのです。こうすると、ベーシックが最後に実行した行番号(つまり、エラーの起こった行番号です)を表示してくれるのです。

```
RUN
      モチデン: 1000

      カケデン: ? 200

Syntax error in 100
Ok
LIST.
100 IF L>K THEN BEEP:GOTO 60
Ok
■
```

●間違いがどこにあるのかわからない場合

エラー表示がしっかり画面に出る場合、デバッグは簡単ですね。エラー表示が出ないのに望みの結果が得られないような場合が一番こまります。

たとえば、変数の値がおかしい場合には、プログラムのどこかに STOP 文をはさんで RUN させます。

STOP 文が実行されると、[CTRL] + [STOP] キーが押されたと同じ状態になります。

ブレーク イン
Break in □□□

(□□□は、STOP命令の行番号です。)

と画面に表示して、プログラムは実行を停止します。そこで、変数の内容を直接 PRINT 文で表示させて調べてみるのです。たとえば、さきほどのルーレット・プログラムの場合に3つの記号がそろっていないのに“オオアタリ!!”が出てしまうときに、

225 STOP

と書いておきます。RUN させれば行番号225で実行は停止しますね。ここで変数Pの値などをPRINTして調べてみるのです。もしPの値が負の数だったりしたらこれはあきらかに間違いですね。つまり行番号225より前のプログラムにバグがあるためにこうなったわけです。このように、プログラムのどこかに STOP 文を書いておくことで、どのへんにバグがあるのかけんとうをつけるのです。

STOP 文で実行を停止しても、

CONT RETURN

としてやれば、実行が再開します。

```
RUN
      モデル： 1000

      カケテン：？ 200

      ***

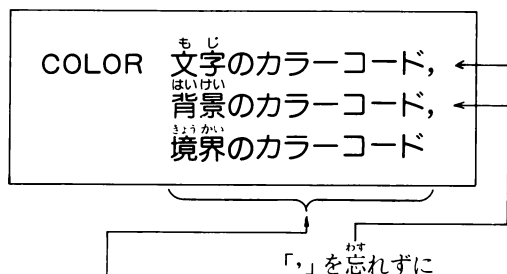
Break in 225
Ok
CONT ←————— とすると実行が再開する。
```

どこにバグがあるかわかったら、あとは、カーソルコントロールキーで修正すれば、デバッグは終了です。
長いプログラムには、ところどころに STOP 文を入れてデバッグしやすくするくふうをしましょう。
もちろんバグがなくなったら、プログラムのなかから STOP 文は消します。

カラー COLOR で色をつけよう

…カラー画面の作り方

画面にたくさん文字や数字を書いてきましたが、全部白一色で、あまりきれいではありません。画面をカラーで使う場合、どのような色を使うかは COLOR 命令で決めます。



0～15の数値、変数、式

• カラーコードは0～15の数値、変数式

カラーコード

0. 透明
1. 黒
2. 緑
3. 明るい緑
4. 暗い青
5. 明るい青
6. 暗い赤
7. 水色
8. 赤
9. 明るい赤
10. 暗い黄
11. 明るい黄
12. 暗い緑
13. 紫
14. 灰
15. 白

●文字の色を変えよう

H 1 の電源を入れた状態ではカラーコードは15（つまり、文字は白）にセットされています。さっそく色を変えてみましょう。

COLOR 1 RETURN

と打ち込んでみましょう。画面全体の文字が黒になりますね。これからはキーを押せば、全部黒で画面に書かれます。一度COLOR命令が実行されると、次に別な色でCOLOR命令が実行されるまで、指定された色は変化しません。もとの白に戻すには

COLOR 15 RETURN

とすればよいのですね。

●背景色を変えよう。

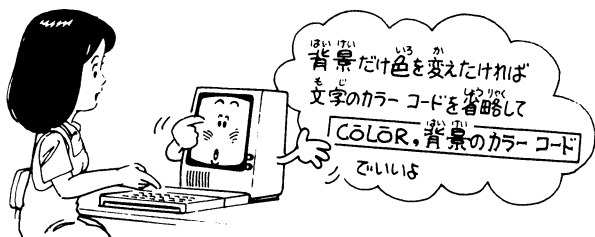
COLOR 命令で背景色を指定することもできます。

H1の電源を入れた状態では背景色のカラーコードは4
(暗い青)にセットされています。背景色を緑にする
ために

COLOR, 2 RETURN

としてみましょう。

背景がきれいな緑色になったでしょう。



●境界色を変えよう。

COLOR 命令で境界色(画面の文字などを表示できる
範囲の外側の色)を指定することができます。

COLOR , , 2 RETURN

としてみましょう。画面全体が背景色と同じ緑色にな
ってしまいましたね。これでは、どこからどこまでに文
字が書けるのかわかりませんね。

●もとの状態に戻すのには、

COLOR15, 4, 7 RETURN

とします。ファンクションキーの F6 を押すと、一
回で入力することができますね。

〈起こりやすいエラー〉

●Illegal function call(イリーガル ファンク
ションコール)

COLOR 命令の後に書くカラーコードが定められた
範囲を越えた場合に起こります。

お絵かきをする前に

スクリーン (SCREEN)

いよいよ次の章から、図形を表示するためのいろいろな命令を説明していくわけですが、お絵かきをする前にどうしてもやっておかねばならない命令があります。それが **SCREEN** 命令なのです。

●SCREEN

H1 は4種類の画面モードを持っています。つまり H1 には今、皆さんが見ている画面のほかに3種類の画面があるのです。

0:テキストモード1

よこ39字×たて24行

1:テキストモード2

よこ29字×たて24行

2:グラフィックモード

ハイリゾリューションモード

よこ256ドット×たて192ドット

3:グラフィックモード

マルチカラー

(ローリゾリューション)モード

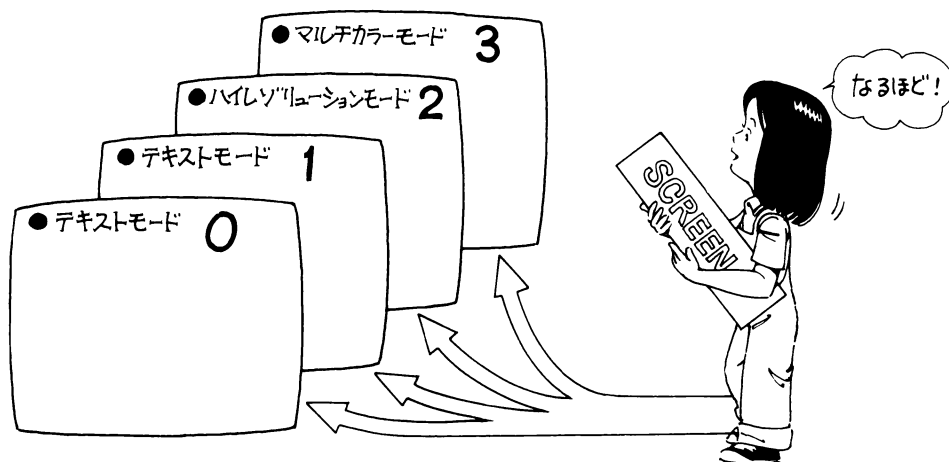
よこ256ドット×たて192ドット

聞いたことのない言葉がならんでいますね。「テキストモード」というのは、画面に文字を表示するモードです。何文字画面に表示するかで、2種類のテキストモードがあるわけです。今まで使ってきた画面はモード1(よこ29字×たて24行テキストモード)なのです。モード2とモード3が、画面に図形をかくときに使うモードです。図形を表示したいときには必ず、モードを2か3にしなければなりません。そのときに使うのが **SCREEN** 命令です。

SCREEN 画面モード

としてやればよいのです。ただし **SCREEN** 命令は、プログラムの中で使わないと意味を持ちません。なぜなら画面モードは、コマンド待ちの状態では必ずテキストモードになってしまうからです。

SCREEN命令は、図形を描くプログラムの先頭に必ず書いてください。そうしないと、次の章で説明する **PSET**、**LINE**、**PAINT** などが、すべてエラーになってしまいます。



SCREEN命令はこの他に、後で説明しますスプライトの大きさや、キーを押したときに出るクリック音、カセットテープのボーレイト (31 ページ参照)、プリンタの有無などを指定するときに使用します。
詳しくは、ベーシック文法編 222 ページをご覧ください。

〈起こりやすいエラー〉

- **Illegal function call** (イリーガル ファンクション コール)

グラフィック命令をテキストモードで実行しようとすると起こります。

H1で絵を描こう

ピーセット ステップ ピーリセット
(PSET, STEP, PRESET)

H1は画面に文字や記号を書くだけではありません。
難しい図形や絵を描くこともできるのです。まずは一番簡単な PSET, PRESET 命令から。

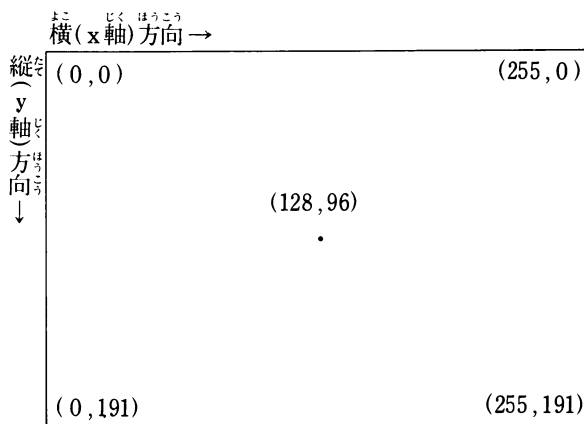
●PSET(ピーセット)

画面に絵や図形を描くためには、細かい点が画面に打てなければなりません。その細かい点をいろいろな形につないでゆくことによって、複雑な絵も描けるわけです。

H1の画面には、横方向に256個、縦方向に192個の点を打つことができます。点の位置は、横(x軸)方向の位置と縦(y軸)方向の位置を

(横方向の位置、縦方向の位置)

というように表します。



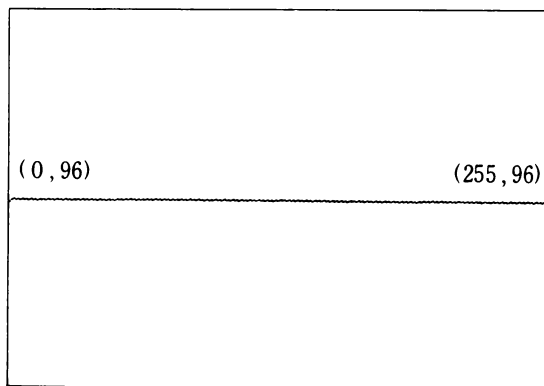
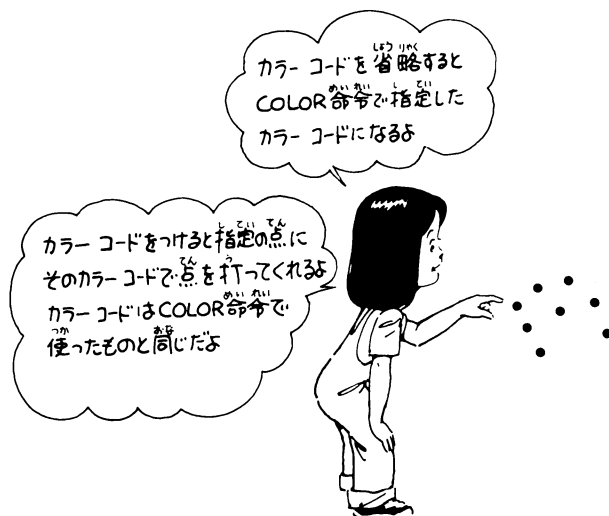
H1はこんなふうに
256(横)×192(縦)個の点を
打つことができるのよ

(0,0) といったら、画面の一番左上の点、(128,96) は画面の真中の点ということになりますね。
さて、ではこの位置に点を打つにはどうしたらよいでしょうか？ そうです。この時に **PSET** 命令を使うのです。

PSET (横方向の位置, 縦方向の位置), カラーコード

```
10 SCREEN 2
20 FOR N=0 TO 255
30 PSET (N,96),1
40 NEXT N
50 GOTO 50
60 END
```

とプログラムして **RUN** させてみましょう。画面中央に黒い水平線が書けました。



黒の直線です

●STEP(ステップ)

画面に最後に打った点から、横方向の変化幅、縦方向の変化幅で、新しく打つ点の位置を指定することができる便利な命令が**STEP**です。

PSET STEP (横方向の変化幅, 縦方向の変化幅), カラーコード

STEP 命令は、**PSET** だけでなく、これから説明する **PRESET, CIRCLE, PAINT, LINE** 命令とも、組み合わせることができます。

ために

```
10 SCREEN 2
20 PSET (128,96),1
30 GOTO 30
40 END
```

と、してみましょう。画面の中央に黒い点が書かれましたね。

CTRL + **STOP** キーでテキスト・モードにもどります。

今度は、**FOR~NEXT** 命令を使って、点をつないで直線を描いてみよう。

●PRESET(ピーリセット)

PRESET 命令は点を消す命令です。

PRESET (横方向の位置, 縦方向の位置), カラーコード

さきほど作ったプログラムに

```
42 FOR J=0 TO 255
44 PRESET (J, 96)
46 NEXT J
```

を追加して、RUN させてみましょう。一度描いた黒い水平線が次に消されていくのがわかりますね。このように、背景色と同じ色で点を消していきます。もし、PRESET に、背景色以外の色を指定すると、PSET と同じです。

〈よく起こるエラー〉

●画面上に何もなくなってしまう。

PSET 命令や PRESET 命令の () の中に、大きすぎる数や負の数を入れた場合に起こります。たとえば、

PSET (-100 , 1500)

のような場合です。[CTRL] キーを押しながら [STOP] キーを押すと、もとの画面にもどります。

注) テレビ信号の特性により指定した色と異なる場合があります。

ちよく せん ちようほう けい かん たん 直線や長方形も簡単に

ライン ペイント (LINE, PAINT)

どんな複雑な図形や絵も、画面上では結局、点の集まりですから、PSET 命令さえあれば、とりあえず何でも画面に描くことができます。しかし、直線1本を描くにも FOR ~ NEXT 命令を使わなければならない、大変です。まして画面上のある部分を塗りつぶす場合など非常に時間もかかることになります。そこで、PSET 命令の機能を拡張したいくつかの命令が用意されています。

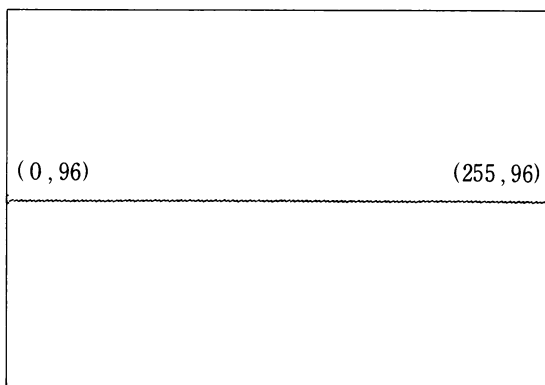


● LINE (ライン)

LINE 命令は、画面のある点からある点までを直線で結ぶ命令です。たとえば

```
10 SCREEN 2
20 LINE (0,96)-(255,96),1
30 GOTO 30
40 END
```

としてみましょう。PSET 命令を使った136ページのプログラムと同じ結果になります。



黒の直線です

こんど
今度は、

```
25 LINE (0,96)-(255,96),4 RETURN
```

としてください。今描いた直線がすぐ消されましたね。
これは今描いた直線の上から背景色と同じ色の直線を
描いただけです。図形を消す時には、背景色と同じ色
を図形の上に塗ればよいわけです。

LINE 命令は

マイナス記号

LINE (始めの位置)-(終わりの位置),
カラー コード

と書くのです。(始めの位置)には、PSET 命令で説明
したように、(横の位置、縦の位置)と、2つの数値を
「,」で区切ってならべます。(終わりの位置)についても
同じです。カラーコードは省略して

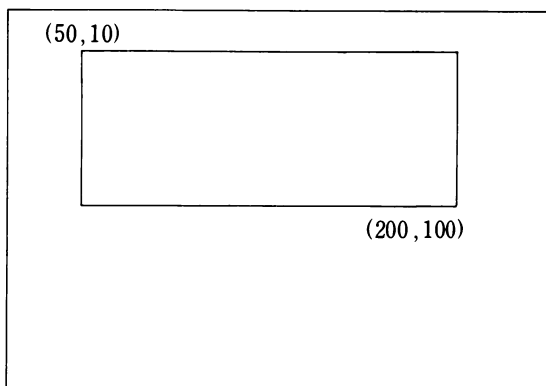
LINE (始めの位置)-(終わりの位置)

としてもかまいません。この場合、COLOR 命令で指
定されている文字のカラーコードが、LINE 命令のカ
ラーコードになります。

●LINE 命令で長方形を描こう

LINE 命令には、もう一つ別な機能があります。それ
は、画面に長方形を描かせる機能です。ために、次
のように打ち込んで、RUN させてみましょう。

```
10 SCREEN 2  
20 LINE (50,10)-(200,100),  
8,B  
30 GOTO 30  
40 END
```



どうです？ 画面に赤い線で長方形が描かれましたね。
今度はちょっと変えて

```
25 LINE (50,10)-(200,100),  
4,B
```

としてRUN させてみましょう。今描いた長方形がす
ぐ消されましたね。これも、背景色と同じ色を長方形
の線に塗っただけです。画面に長方形を描く時は

LINE (始めの位置)-(終わりの位置),
カラー コード, B

とすればよいのです。

線を引くだけの LINE 命令とちがうのは、一番最後の「B」です。この「B」を LINE 命令の最後に付けることで、画面に長方形を描く命令になります。また、LINE 命令で長方形の中を塗りつぶすこともできます。行番号25を消して

```
20  LINE (50, 10) - (200, 100),  
    8, BF
```

としてみましょう。長方形を描くのはさっきと同じですが、長方形の中も赤で塗りつぶしていますね。次に

```
25  LINE (50, 10) - (200, 100),  
    4, BF
```

とすれば、赤で塗られた長方形が消えるのはもうわかりますね。

LINE (始めの位置) - (終わりの位置),
 カラー コード, BF

このように、一番最後に、「B」のかわりに「BF」とすれば、長方形を塗りつぶす命令になるわけです。

●PAINT (ペイント)

PAINT 命令は、線で囲まれた内側や外側を好きな色で塗りつぶす命令です。長方形の内側を塗りつぶす場合は LINE で簡単にできるのですが、その他の複雑な図形を塗りつぶす場合は困りますね。それでこの命令があるのです。

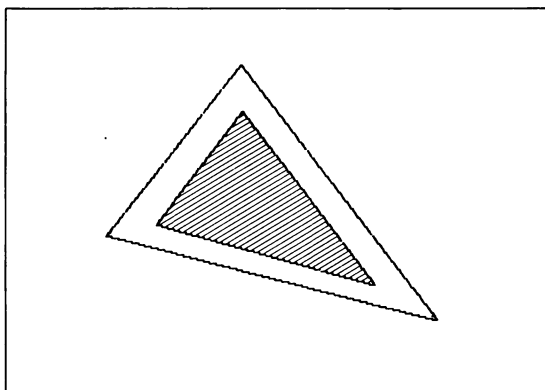
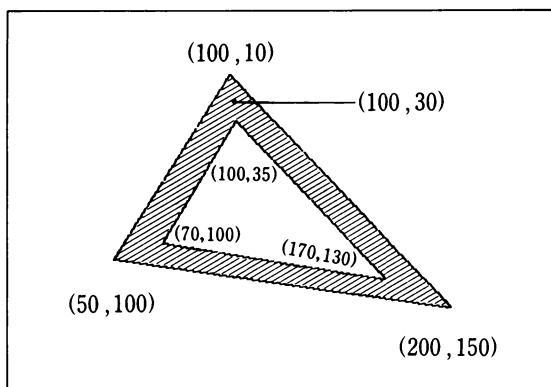
次のプログラムを RUN させてみましょう。

```
10  SCREEN 2  
20  LINE (100, 10) - (50 100), 1  
30  LINE  - (200, 150), 1  
40  LINE  - (100, 10), 1  
50  LINE (100, 35) - (70, 100), 1  
60  LINE  - (170, 130), 1  
70  LINE  - (100, 35), 1  
80  PAINT (100, 30), 1, 1  
90  GOTO 90  
100 END
```

ここで、行番号30, 40, 60, 70は、LINE命令の「(始めの位置)」の部分省略していますがこの場合、前に実行したLINE命令の「(終わりの位置)」がそのまま「(始めの位置)」になります。

黒の三角形を大少2つ描き、外側と内側の間を黒く塗りつぶしましたね。

PAINT (100,30),1は(100,30)の位置から黒(1)で塗りつぶしなさいの意味です。



今度は内側の三角形の内部を黒く塗りつぶしますね。

PAINT(100,100),1は(100,100)の位置から、とにかく囲まれている部分を黒(1)で塗りつぶしなさいの意味です。

80 PAINT (0,0),1

としたらどうなるでしょうか。RUNしてみてください。

PAINT (塗り始めの位置), 塗る色,
境界線の色

「塗り始めの位置から、その色の境界線で囲まれている部分を、塗りつぶしなさい」というのがPAINT命令の意味です。

「境界線の色」を省略することができます。そうすると、「とにかく線に囲まれた部分を塗りつぶしなさい」という命令になります。

ここで使用しているハイリゾリューションモード (SCREEN2)では、塗る色に境界線の色を指定しなくてはなりません。

マルチカラーモード (SCREEN3)では、塗る色と、境界線の色を別々に指定できます。

さっきのプログラムの行番号 80を

80 PAINT (100,100),1

と換えてRUNしてみましょう。

PAINT命令で注意することは、囲んでいる線が少しでもとぎれてはいけな、ということです。とぎれている部分から色もれてしまいます。

<よく起こるエラー>

● Illegal function call (イリーガル ファンクション コール)

PAINT命令に使う数値が、規定外の場合です。

PAINT (100,100),100

などが、この種のエラーを起こします。この場合は、カラーコードが100で大きすぎます。

サークル えん えが CIRCLE で円も描けるぞ

点の打ちかた、線のひきかた、長方形の描きかたはもう
おわかりですね。今度は CIRCLE 命令を使って、円や
だ円を描いてみましょう。

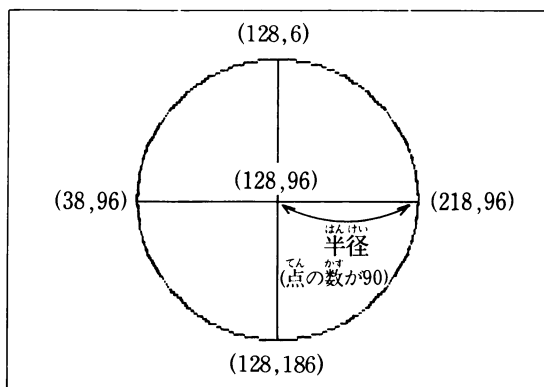
●CIRCLE (サークル)

CIRCLE(中心の位置), 半径,
カラー コード

これで円が描けるはずですが。(中心の位置) には PSET
命令で説明したように (横の位置、縦の位置) と 2 つ
の数値を「,」で区切って並べます。半径は点の数です。
さっそくやってみましょう。

```
10 SCREEN 2
20 CIRCLE (128, 96), 90, 15
30 GOTO 30
40 END
```

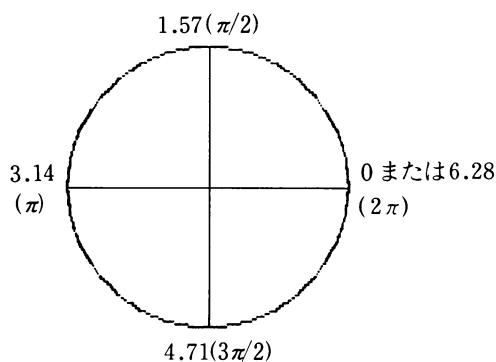
と打ち込んで RUN させてください。画面に大きな、
白い円が描けますね。



CIRCLE 命令の機能はこれだけではありません。だ円、
孤、円に関するものなら何だって描けるのです。

CIRCLE (中心の位置), 半径, カラー
コード, 開始角度, 終了角度, 比率

- 比率を書くとか円を描きます。1を中心それぞれ
りも、小さければ平たいだ円、大きければ縦長の
だ円。
省略すると、1とみなし円を描きます。
- 開始角度は円弧を描き始める位置を、終了角度は
描き終る角度を0~6.28とラジアンで指定します。
(下図)

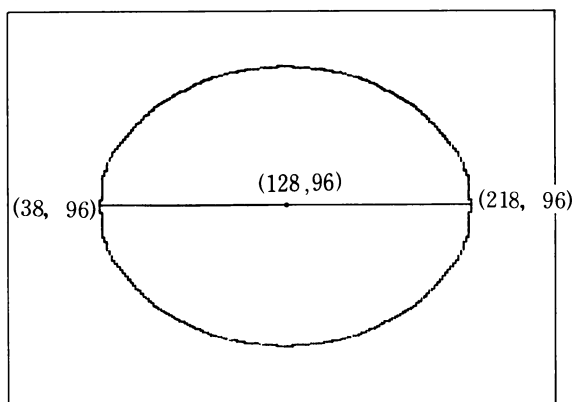


開始角度を省略すると0、終了角度を省略すると
6.28とみなされます。

ただ読んだだけではわかりづらいですね。実際にいろい
ろやってみましょう。さきほどのプログラムの行番号20を

```
20 CIRCLE (128, 96), 90, 15, ,,  
0.7
```

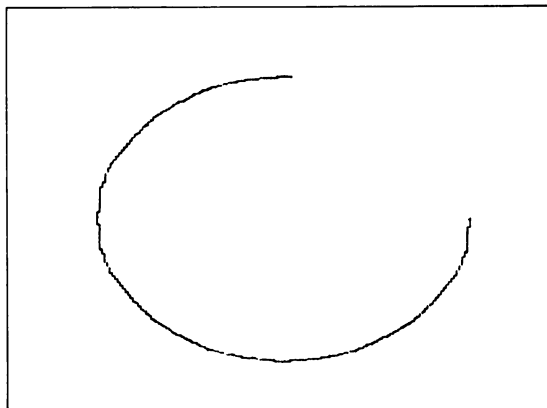
として RUN させてみます。
これで白いだ円が描けます。途中の機能を省略する場
合でも、「,」だけは書いてください。



次に行番号20を

```
20 CIRCLE (128, 96), 90, 15,  
1.57, 6.28, 0.7
```

として RUN させてみましょう。右上1/4が欠けた白
色の横長のだ円が描かれますね。



スプライトで図形を動かそう

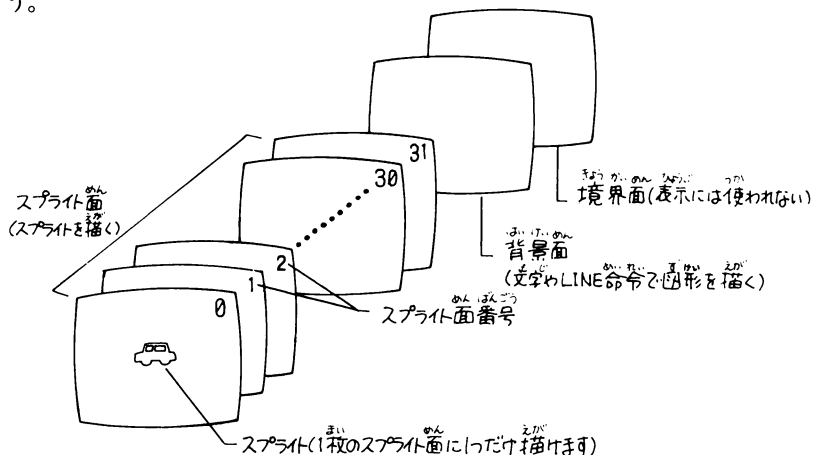
スプライト プット スプライト
(SPRITE\$, PUT SPRITE)

PSET, LINE, PAINT, CIRCLE と、図形を描くための色々な命令を学んできました。でも、こういった命令で作った図形を動かすのはなかなかめんどろなことです。たとえば、LINE 命令、PAINT 命令を10回使って作った図形を動かすことを考えてみましょう。動かすたびに、やはり10回ぐらいこれらの命令を使わなければなりませんね。これでは、プログラムを書くのも大変ですし、図形を描くのに時間もかかって、速い動きをするゲームなど、とても作れそうにありません。

こんなときには、スプライトが便利なのです。好みの図形を簡単に作れて、しかも、とっても速く画面を動かすことができるのです。

●スプライトって何？

スプライトを学ぶ前に、H 1の画面についてちょっと説明しておきましょう。



H 1の画面はスプライト面、背景面、境界面と大きく3つに分けることができます。境界面は画面表示には使われないので、ここではスプライト面と背景面について説明しましょう。

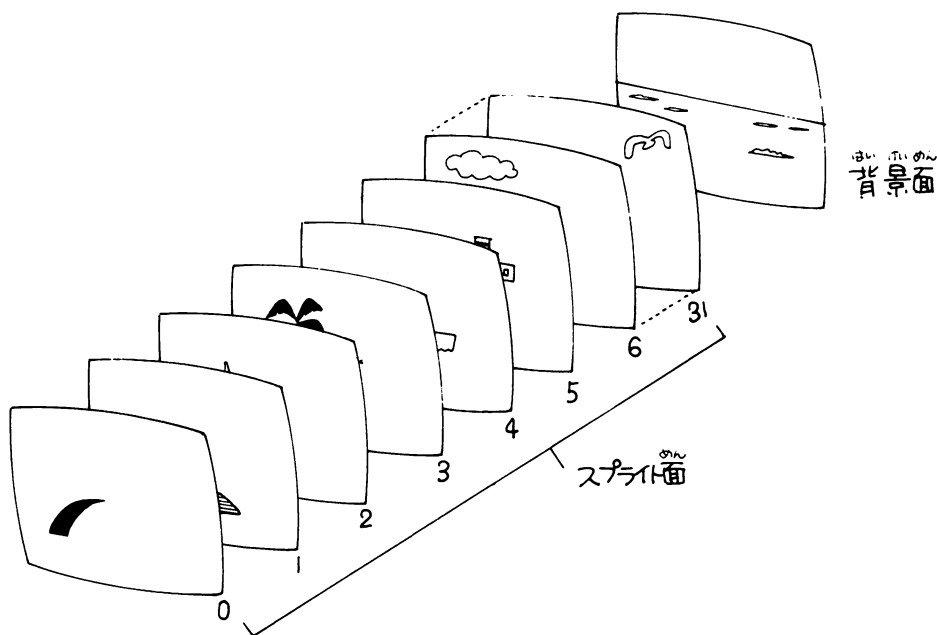
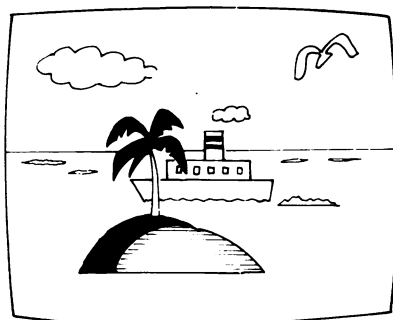
まずは背景面から。今まで説明しませんでした、PRINT 命令による文字の表示や、PSET, LINE, PAINT, CIRCLE などの図形表示は、すべてこの画面上で行なわれてきたのです。これらの図形を表示させる命令は、さきほど説明したように、図形を動かす目的にはほとんど使えません。つまり、動かない図形に使う画面なので、背景面と呼ぶわけです。

それでは、スプライト面はどんな図形に使われるか、もうおわかりですね。そうです。図形を動かす目的で使うのが、このスプライト面なのです。

もう少し、スプライト面について説明しましょう。スプライト面には0から31までのスプライト面番号がつけられており、番号の小さい方から優先的に表示されるようになっていきます。この面、1枚1枚に、好きな図形を1個だけ書き込むことができます。ただし、スプライト面が異なれば、同じ図形を32個まで表示できます。スプライト面は、何も描かれていなければ透明なので、テレビの画面には32枚分重なって見えることになります。また、このために、後方にあるスプライト面に描かれた図形は、前方のスプライト面に描かれた図形と重なると、見えなくなってしまう。だから奥行のある画面ができるわけです。

さあ、それではスプライトについて説明しましょう。スプライトというのは、32あるスプライト面に書き込む図形のことなのです。スプライトの書き方は、LINE命令などで図形を作るときとは、ちょっと違ったやり方をします。次のSPRITE \$ で説明しましょう。

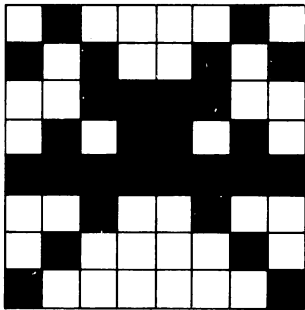
実際の画面では
こんなふうに見えるよ



●SPRITE\$ (スプライト・ドル)

スプライトを作る命令が、このSPRITE\$ です。スプライトは図形を点の集まりで作ります。まずは、点か縦横それぞれ8×8ドットの大きさのスプライトを作る場合について説明しましょう。

まず、8×8ドットのます目を作って、そこに図形を描いてください。空白を0、黒く塗ったところを1として、2進数で表現します。



2進数	16進数	10進数
&B01000010	=&H42	= 66
&B10100101	=&HA5	= 165
&B00111100	=&H3C	= 60
&B01011010	=&H5A	= 90
&B11111111	=&HFF	= 255
&B00100100	=&H24	= 36
&B01000010	=&H42	= 66
&B10000001	=&H81	= 129

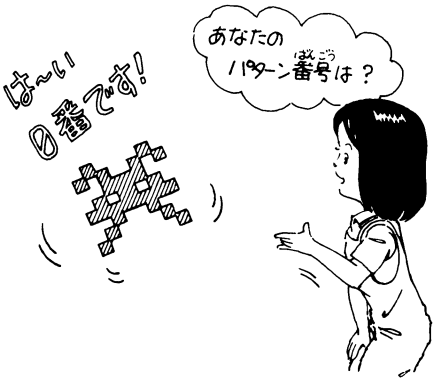
こうしてできた、8つのデータをキャラクタコードとする文字列が、スプライトを表します。つまり、図のような図形をスプライトに設定したい場合、

```
CHR$(66)+CHR$(165)+CHR(60)+CHR$(90)
+CHR(255)+CHR$(36)+CHR$(66)+CHR$
(129)
```

という文字列で表現するわけです。もちろん、10進数で書かずに、そのまま、2進数や16進数を使って

```
CHR$(&B01000010)+CHR$(&B10100101)+...
CHR$(&B01000010)+CHR$(&B10000001)
CHR$(&H42)+CHR$(&HA5)+...+CHR$(&
H42)+CHR$(&H81)
```

としてもよいわけです。



スプライトが8×8ドットでできている場合、H 1は256個の異なるスプライトを持つことができます。それぞれスプライト番号で区別します。これをスプライトのパターン番号といいます。ですから、スプライトを作る場合は、図形を表す文字列を作っただけではダメで、パターン番号も決めてやらなければなりません。

それでは、さきほど作った図形（文字列で表されている）をパターン番号0のスプライトとしてみましょう。

```
SPRITE$(0)=CHR$(66)+CHR$(165)+CHR$(60)+CHR$(90)+CHR$(255)+CHR$(36)+CHR$(66)+CHR$(129)
```

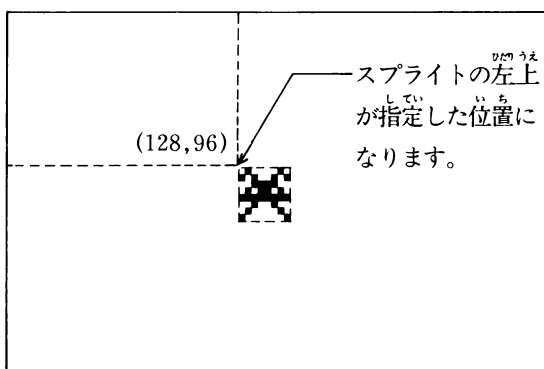
SPRITE\$ のあとにかっこでくくってパターン番号を書き、次に「=」のあとに、図形を表す文字列を書いてやればよいわけです。

SPRITE\$(パターン番号)=図形を表す文字列

さあ、これでスプライトは完成しました。さっそく作ったスプライトを画面に表示しましょう。それには、PUT SPRITE 命令を使いますが説明は後にして、まず、次のプログラムを打ち込んでRUNさせてください。

```
10 SCREEN 2,0
20 FOR N=1 TO 8
30 READ A:B$=B$+CHR$(A)
40 NEXT N
50 SPRITE$(0)=B$
60 PUT SPRITE 0,(128,96),8,0
70 GOTO 70
80 END
90 DATA 66,165,60,90,255,36,66,129
```

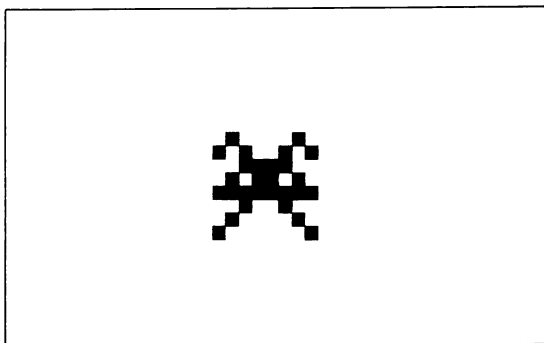
さっき描いた図形が赤で表示されましたね。



図形が小さくても構いませんか？。そんな時は、今のプログラムの行番号10を次のように換えてください。

10 SCREEN 2,1.

さあ、RUN してみましょう。さっきのちょうど2倍の大きさの図形を描きましたね。これは8×8ドットのスプライトの1点1点を2倍に拡大しているのです。大きくはなりましたが、少し荒い図形になりましたね。



●PUT SPRITE (プット スプライト)

作ったスプライトを画面に表示する命令です。147ページのプログラムの行番号60をよく見てみましょう。

```
60 PUT SPRITE 0,(128, 96),8,0
```

スプライト面番号
画面位置
カラーコード
パターン番号

この命令は、「スプライト面番号0の画面の、画面位置(128, 96)に、カラーコード8(赤)で、パターン番号0のスプライトを描きなさい」という意味なのです。

PUT SPRITE スプライト面番号, 画面位置, カラーコード, パターン番号

スプライト面番号はもう説明しましたね。0～31の値がとれます。画面位置(128, 96)という書き方ももうだいじょうぶですね。最初に横方向の位置、次に縦方向の位置を書くのですよ。パターン番号のところには **SPRITE\$** 命令で作ったスプライトのパターン番号を書きます。 **PUT SPRITE** を実行する前に、**SPRITE \$** でスプライトを作っておかないと、画面には何も書きませんので注意してください。

今度は147ページのスプライトを、実際に動かしてみよう。

```
10 SCREEN 2,1
20 FOR N=1 TO 8
30 READ A:B$=B$+CHR$(A)
40 NEXT N
50 SPRITE$(0)=B$
60 PUT SPRITE 0,(0,80),1,0
62 C=INT(RND(1)*15)
64 FOR I=1 TO 256
66 PUT SPRITE 0,STEP(1,0),C,0
```

68 NEXT I

70 GOTO 62

80 END

90 DATA 66,165,60,90,255,36,66,129

書き換えたところや、新しくつけ加えた行には下線が引いてあります。カーソルキーをうまくつかって修正してください。できたら、さっそく **RUN** してみましょう。



さきほど作ったインベーダーのような図形が画面の左から右へ動いていますね。

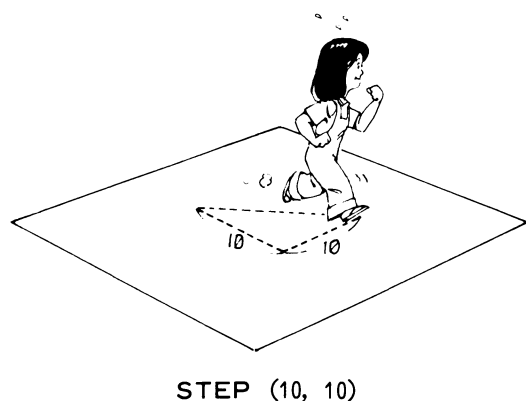
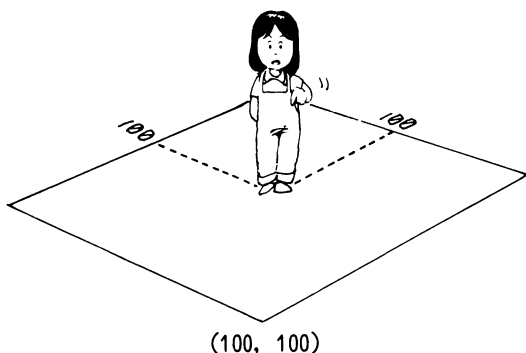
[STOP] キーを押すと、止まったりまた動き出したりします。

PUT SPRITE 命令で、画面位置を指示するところに

STEP (横方向の変化幅, 縦方向の変化幅)

と置くと、「指定したスプライト面の以前の位置から、横方向の変化幅、縦方向の変化幅だけスプライトを動かさない」という意味になるのです。スプライトを描くだけなら（横方向の位置、縦方向の位置）と書けばよいのですが、スプライトを動かしたい場合は、STEPを使います。

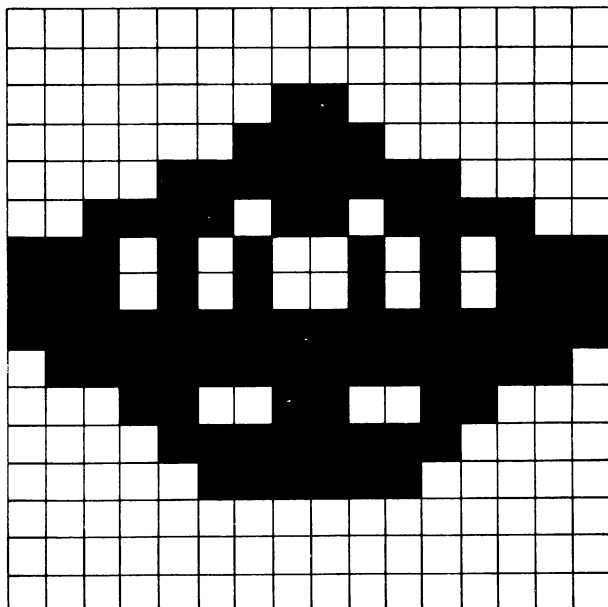
(前で説明したPSET, PRESET, LINE, PAINT, CIRCLE命令もこのSTEPが使えます。)



●もっと大きなスプライトを書いてみよう

8×8ドットのスプライトを拡大して表示する方法は説明しましたね。今度は、最初から16×16ドットのスプライトを作る方法を説明しましょう。

16×16ドットのます目を留意して、そこに図形を描いてください。空白を0、黒く塗ったところを1として2進数になおすまでは、前と同じですね。



0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 1	1 0 0 0	0 0 0 0
0 0 0 0	0 0 1 1	1 1 0 0	0 0 0 0
0 0 0 0	1 1 1 1	⋮	⋮
0 0 1 1	1 1 0 1	⋮	⋮
1 1 1 0	1 0 1 0	⋮	⋮
1 1 1 0	1 0 1 0	⋮	⋮
1 1 1 1	1 1 1 1	⋮	⋮
⋮	⋮	⋮	⋮



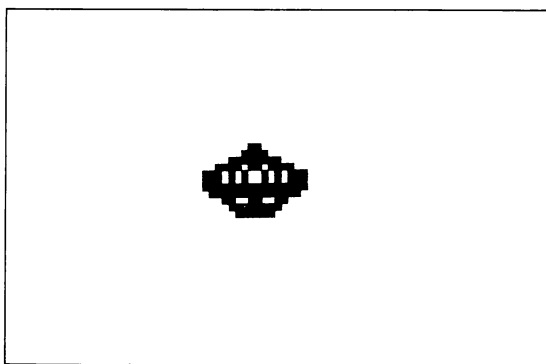
できた2進数をキャラクタコードになおして、文字列にしていく点も前と同じです。ただ、ちょっと気をつけなければならないのは、文字列にしていく順序です。図に書いたように左上の2進数から順に、文字列に入れてゆくのです。

さあ、それでは、さっそく作って動かしてみましょう。

```
10 SCREEN 2,3
20 FOR I=1 TO 32
30 READ A:B$=B$+CHR$(A)
40 NEXT I
50 SPRITE$(0)=B$
60 PUT SPRITE 0,(110,0),1,0
70 C=INT(RND(1)*15)
80 FOR I=1 TO 70
90 PUT SPRITE 0,STEP(0,1),C,0
100 NEXT I
110 GOTO 70
120 DATA &B00000000,&B00000000.
    &B00000001,&B00000011,&B00001111,
    &B00111101,&B11101010,&B11101010
130 DATA &B11111111,&B01111111,
    &B00011001,&B00001111,&B00000111,
    &B00000000,&B00000000,&B00000000
140 DATA &B00000000,&B00000000,
    &B10000000,&B11000000,&B11110000,
    &B10111100,&B01010111,&B01010111,
150 DATA &B11111111,&B11111110,
    &B10011000,&B11110000,&B11100000,
    &B00000000,&B00000000,&B00000000
160 END
```

120のDATAが左上¼、130のDATAが左下¼、140のDATAが右上¼、150のDATAが右下¼を示しています。

行番号10でSCREEN2,3としている点に注意してください。スプライトサイズを16×16に設定していますね。こうしないと、8×8の普通のスプライトを描いてしまいます。16×16ドットのスプライトは全部で64個までつくれます。また、行番号120からのDATA文が少々こみいっています。これはデータを2進数で書いたためですが、これらを10進や16進になおしてプログラムしてもかまいません。少々手間がかかりますが…。さあ、RUN してみましょう。



大きなUFOが色を変えながら画面の上から下へ動いていますね。

文字と数字を交換しよう

…VAL関数, STR\$関数の使い方

文字列と数はまったく別のものですから組み合わせて計算することはできないと説明しました。

次のように打ち込んでみましょう。

```
A$="123" RETURN
```

では、A\$は123という数ではなく、「123」という文字列なのです。こうした文字列を数値に、また数値を文字列に換えることができます。H 1を時計にして使うときなどにきっと便利ですよ。

そんなときに使うのが VAL と STR\$ です。

```
10 A$="12345":B$="54321"
20 C$=A$+B$
30 PRINT C$
40 C=VAL(A$)+VAL(B$)
50 PRINT C
60 END
```

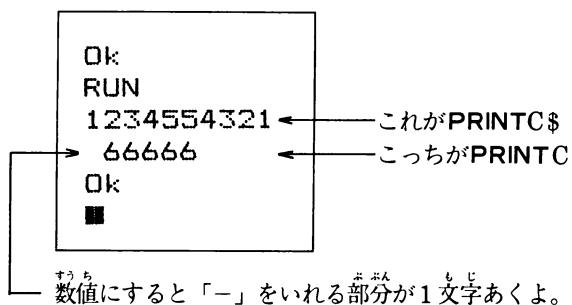
●VAL (VAL)関数

VAL を使うと数字で書いてある文字列を数値にすることができます。

VAL (文字列か文字変数)

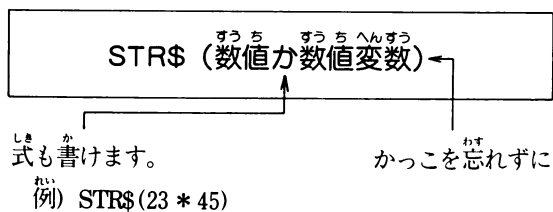


さっそく RUN してみましょう。図のような画面になりましたね。行番号20の「+」は、たし算の「+」であることに注意してください。

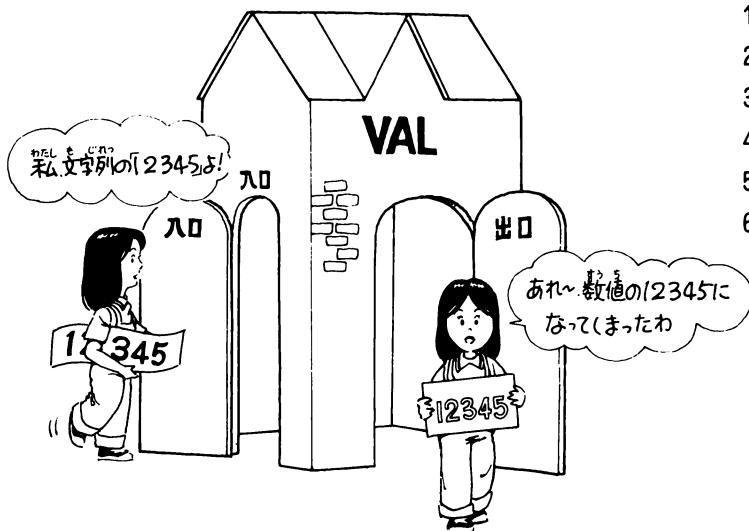


●STR\$ (ストリング ドル) 関数

このSTR\$関数はVAL関数とはまったくの反対で、数値を文字列に換える働きをします。

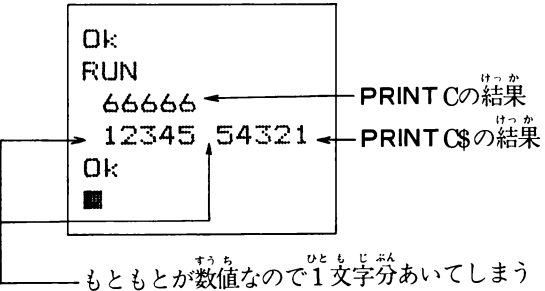


次のプログラムを打ち込んで、RUN させてみましょう。

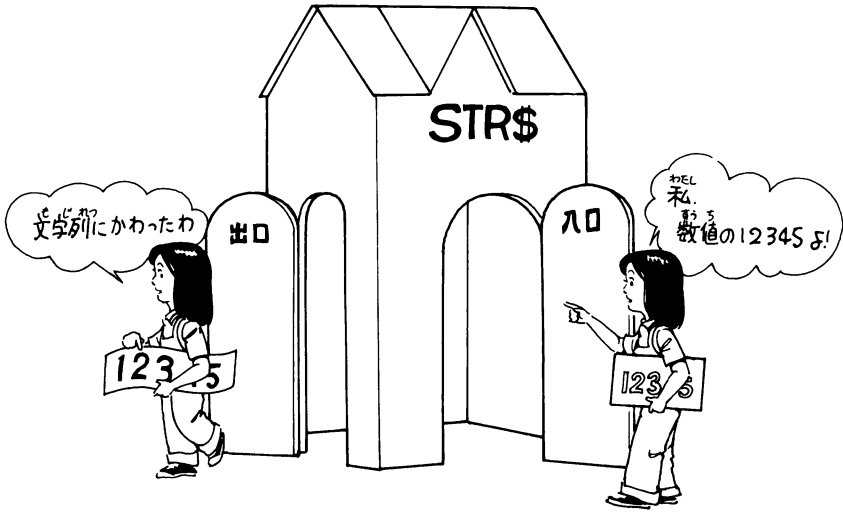


```
10 A = 12345 : B = 54321
20 C = A + B
30 C$ = STR$(A) + STR$(B)
40 PRINT C
50 PRINT C$
60 END
```

行番号30の「+」は、計算するときのたし算の記号ではなくてここでは文字列をつなげる「+」であることに注意してください。



〈起こりやすいエラー〉
●Type mismatch(タイプ ミスマッチ)
108 ページ、314 ページをよく読んでください。



キャラクターコードのお話 はなし

キャラクターコード かん すう アスキー かん すう (CHR\$関数, ASC関数)

H 1では、たくさんの文字をキーボードから入力して表示することができますが、コンピュータには、キーボードにある以外の文字や図形も表示させることができるのです。

文字や図形に番号を割りあてた表がコンピュータの内部にあるのです。この、割りあてられた番号を、キャラクターコードといいます。

Qは $5 \times 16 + 1 = 81$ となります。

ただし、左の表（2バイトコード）の場合は、次のCHR\$（キャラクターコード）関数をご覧ください。

キャラクターコード表 ひょう

2バイトコード
(上位1バイト01H)

1バイトコード

		上位4ビット		上位4ビット															
		4	5	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
下位4ビット	0		π		0	@	P	'	p	♠			一	タ	ミ	た	み		
	1	月		!	!	A	Q	a	q	♥	あ	。	ア	チ	ム	ち	む		
	2	火		"	2	B	R	b	r	♣	い	「	イ	ツ	メ	つ	め		
	3	水		#	3	C	S	c	s	♦	う	」	ウ	テ	モ	て	も		
	4	木		\$	4	D	T	d	t	○	え	、	エ	ト	ヤ	と	や		
	5	金		%	5	E	U	e	u	●	お	・	オ	ナ	ユ	な	ゆ		
	6	土		&	6	F	V	f	v	を	か	ヲ	カ	ニ	ヨ	に	よ		
	7	日		'	7	G	W	g	w	あ	き	ア	キ	ヌ	ラ	ぬ	ら		
	8	年		(8	H	X	h	x	い	く	イ	ク	ネ	リ	ね	り		
	9	円)	9	I	Y	i	y	う	け	ウ	ケ	ノ	ル	の	る		
	10	時		*	:	J	Z	j	z	え	こ	エ	コ	ハ	レ	は	れ		
	11	分		+	:	K	[k]	お	さ	オ	サ	ヒ	ロ	ひ	ろ		
	12	秒		,	<	L	¥	!		や	し	ヤ	シ	フ	ワ	ふ	わ		
	13	百	大	-	=	M]	m	!	ゆ	す	ユ	ス	ヘ	ン	へ	ん		
	14	千	中	・	>	N	^	n	~	よ	せ	ヨ	セ	ホ		ほ			
	15	万	小	?	0	-	o			っ	そ	ッ	ソ	マ		ま			

コンピュータはこのキャラクターコードの表から「A」という文字は65番、「!」という記号は33番というようにさがしてきて表示しているのです。

キャラクターコードを扱う命令に、CHR\$ と、ASCがあります。

●CHR\$（キャラクターコード）関数 かん すう

CHR\$ は、キャラクターコードで文字を探す関数です。

CHR\$（数値か数値変数）



キャラクターコードは上の表から次のような式で求めます。

(横方向の数字) × 16 + (縦方向の数字) = (求めるキャラクターコード)

たとえば

PRINT CHR\$ (65) RETURN

と打ってください。「A」を画面に表示しますね。キャラクターコード表の65番目の文字はAであったわけです。


```
Ok
PRINT CHR$(65)
A
Ok
■
```



(65) を他の数値に変えていろいろな文字や図形を表示させてみましょう。

2 バイトコード (グラフィック文字) の場合は、

```
CHR$(1)+CHR$(数値か数値変数)
```

例えば、月

CHR\$(1)+CHR\$(4×16+1) です。

この CHR\$ 関数を使って、キャラクターコード表を画面に書いてみましょう。

```
10 CLS
20 FOR X=2 TO 15
30 FOR Y=0 TO 15
40 LOCATE X,Y
50 IF X=7 AND Y=15 THEN
    PRINT CHR$(32):GOTO70
60 PRINT CHR$(X*16+Y)
70 LOCATE 0,Y
80 PRINT CHR$(1)+CHR$(4
    *16+Y)
90 LOCATE 1,Y
100 PRINT CHR$(1)+CHR$(5
    *16+Y)
110 NEXT Y
120 NEXT X
130 END
```

このプログラムを RUN させてください。16×16で256個の文字や記号を画面に表示します。

注) キャラクターコードの1～31と127は、コントロールコードに使用されています。詳しくは、311ページのコントロールコード一覧表をご覧ください。

●ASC (アスキー)関数

ASC は CHR\$ とは逆に、文字からキャラクターコードを探し出す関数です。

ASC (文字列か文字変数)



```
PRINT ASC ("A") RETURN
```

と打ってください。「A」のキャラクターコード65を画面に書いてくれます。

また、

```
PRINT ASC ("ABC") RETURN
```

などとかこの中に文字列を書いた場合、「ABC」の先頭の文字「A」のキャラクターコードを表示します。

この場合、65ですね。

ただし、2バイトコードのキャラクターコードの場合には、すべて1を返します。

```
Ok
PRINT ASC ("A")
65
Ok
PRINT ASC ("ABC")
65
Ok
■
```

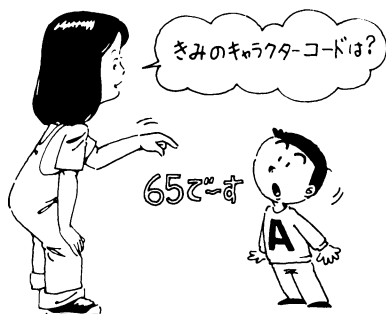
〈起こりやすいエラー〉

●Illegal function call(イリーガル ファンク ション コール)

0～255範囲外の数を使った場合に起こります。

CHR\$ (300)

などとした場合がそうです。



H1を時計にしよう

…時計機能とは

タイム
(TIME)

H1は文字を書くことができました。絵も描けました。
それだけではありません。H1は時計にもなるのです。
このとき使うのがTIMEです。

●TIME (タイム)

TIME は60倍の秒単位で教えてくれるシステム変数
です。次のように打ち込んでください。

```
PRINT TIME RETURN
```

どんな結果が出ましたか？ この結果は、H1の電源
を入れてからどのくらい時間がたったかを60倍の秒で
示しているのです。

```
Ok  
PRINT TIME  
652  
Ok  
■
```

ここは、ほかの表示がでて
も間違いではありません。

TIME の値は、あなたが自由に変えることができます。
つまり、時計の時刻をセットするように、TIMEを変
えることができるのです。

たとえば、

```
TIME=0 RETURN  
PRINT TIME RETURN  
PRINT TIME RETURN
```

⋮

とすると、TIME=0にした時点からどのくらい時間
がたったかを教えてくれます。だから実際の秒は、60
で割れば良いのです。

```
TIME=0
```

```
Ok  
PRINT TIME  
360  
Ok  
PRINT TIME  
10800  
Ok
```

ここまでで60秒たった
ことを意味する
ここまでで180秒たった
ことを意味する

ここはほかの表示がでて
も間違いではありません

今度は、この TIME を使って、簡単な時計を作ってみ
ましょう。

```
10  CLS
20  TIME=0
30  LOCATE 0,10
40  PRINT SPC(6); "分"; SPC
    (6); "秒"
50  S=INT (TIME/60) MOD 60
60  M=INT (TIME/3600)
70  LOCATE 2,10:PRINT M
80  LOCATE 9,10:PRINT S
90  GOTO 50
100 END
```

RUNさせると、そのときからの経過時間を

例

11 分 18 秒

← ここは、ほかの表示がでて
も間違いではありません。

とあなたに教えます。

ただし、このプログラムでは、18分12秒
までの表示で、これを過ぎると0分0秒
になり、もう一度最初からカウントを始
めます。

H1は音だって出せるんだ

ビープ プレイ (BEEP, PLAY)

H1は画面に絵や文字を書くだけではありません。音楽を演奏させることもできるのです。まずは、音を出すだけのBEEPから。

●BEEP(ビープ)

BEEPは、ブザーを鳴らす命令です。

BEEP (RETURN)

としてみてください。「ピーッ」とブザーが鳴りましたね。今度は

```
10 FOR N=1 TO 10
20 BEEP
30 NEXT N
40 END
```

とプログラムを作ってRUNさせてみましょう。「ピーッ、ピーッ……」と10回ブザーが鳴りますね。

このBEEPは、音の高さも音の長さも変えることができないので、音楽演奏には向いていません。H1で音楽を演奏させたい人は、次のPLAY命令をお使いください。

●PLAY(プレイ)



PLAY "CDEFRGAB" (RETURN)

と打ち込んでみましょう。まず、ドレミファと音を出し、少し休んだあとソラシと演奏しますね。この「A」～「G」の文字は、それぞれ

文字	C	D	E	F	G	A	B
音階	ド	レ	ミ	ファ	ソ	ラ	シ

というふうに音階を表わしているのです。また「R」は休符を意味します。

このように、「A」～「G」までの文字や「R」を「」でかこんでPLAYの後に書くと、曲が演奏できるのです。

PLAY "曲を表わす文字列"

今度は、音の長さを変えてみましょう。それには「A」～「G」、「R」のすぐ後に音の長さを表わす数字を書けばよいのです。

PLAY "C1D2E4F8R1G16A32B64"
RETURN

ドレミファと、だんだん音を短かくしていき、しばらく休んだのちソラシと、もっと短い音で演奏していくのがわかりますね。音の長さは1～64まで細かく指定することができます。ただし、4が4分音符、8が8分音符というように、実際の音符に対応しているのは1、2、4、8、16、32、64だけです。数が大きくなると音が短くなることに注意してください。

さて、それでは符点4分音符を出すときにはどうすればよいでしょうか？

PLAY "C4." RETURN

というふうに、数字の後に「.」（ピリオド）をつけてしまうのです。

ここまでで、音楽演奏のための基本的なことはほぼ説明しました。PLAY命令はもっと簡単に、そしてもっといろいろな音を出すために、このほかにもいろいろな機能を持っているのです。

イ) #、+、- ……シャープ、フラット

曲を演奏するためには半音上げる（シャープ）ことや、半音下げる（フラット）ことも必要ですね。

#、+：「A」～「G」のすぐ後につけると半音上げた音を出す。

-：「A」～「G」のすぐ後につけると半音下げた音を出す。

「#」、「+」、「-」はそれぞれピアノの黒い鍵盤に当たります。ただし、例外としてC-はB、E+はF、F-はE、B+はCとなって現在指定しているオクターブの音となります。

たとえば4分音符のシャープの音を出すときは

PLAY "C+4" RETURN

とすればよいのです。「#」、「+」、「-」は「A」～「G」と音の長さの間に入れるということを覚えておいてください。

□) L…音の長さの指定

今までは、「A」～「G」の後に数字を書いて、音の長さを指定しましたね。でもそれではちょっとめんどうなこともあります。全部同じ音の長さで曲を演奏したいときなどがそうです。

たとえば

PLAY "C4D4E4F4G4A4B4"
RETURN

としたい場合などです。全部に4分音符を指定するのはめんどうですね。

「A」～「G」、「R」の後に数字を書かないでよくと、ある決まった長さの音を出すことができます。その長さを指定するときに「L」という文字を使います。さっきの例を使うと

PLAY "L4CDEFGAB" RETURN

とすればよいのです。

つまり、音の長さには「音階を表わす文字の次に数字があれば、その長さで音を出す。なければ、「L」の後に書いた数字の長さで音を出す。」という規則があるのです。この「L」は一度指定すると、次の「L」がくるまで、ずっとそのままになっています。たとえば「L8」と指定したら、電源を切るまで、「A」～「G」、「R」の後に数字のないものは8分音符で音を出すというわけです。また、電源を入れたときは「L4」を指定したと同じ状態になっています。

ハ) O…オクターブの指定

これまでの説明では、1オクターブぶんの音しか出すことができません。もっと高い音や低い音を出すにはどうしたらよいでしょうか？
それには「O」と数字を音階を表わす文字の前に入れます。

```
PLAY "O3CDEFGABO4CD  
EFGABO5CDEFGABO4"
```

RETURN

としてみましょ。3オクターブ分ドレミファソラシと演奏されたでし。 「O」のすぐ後に書いた数字でオクターブを指定することができるのです。数字は1～8、つまり8オクターブの音をH1は出せるのです。 「O」も一度指定すると、次の「O」までその値を保ちます。また、電源を入れたときは「O4」のオクターブになっています。
ここまで説明した命令を使って、曲を1つ作ってみましょ。

```
10 FOR N=0 TO 3  
20 PLAY "L4C.L8DL  
4E.L8CL4  
ECL2E"  
30 NEXT N  
40 END
```

ここで、C.E.は、それぞれC4.E4.と同じです。

二) T…テンポの指定

曲全体を速く演奏したり、遅く演奏したりするときには「T」という文字を使います。さっき作ったプログラムの20行目を次のように書き換えましょ。
T240を追加するだけです。

```
20 PLAY "T240L4C.L8DL4E.  
L8CL4EC L2E"
```

これでRUNさせてましょ。さっきの2倍の速さで演奏していますね。

「T」のすぐ後の数字で、曲全体の速さを変えられるわけです。数字は32～255の範囲で使えます。数字が大きい方が速く演奏します。また、この「T」も次の「T」がくるまで、ずっとそのままになっています。電源を入れたときは「T120」の速さと同じです。

ホ) V…ボリューム

次は音の大きさも変えてましょ。それには「V」と数字を音階を表わす文字の前に入れます。

```
PLAY "L1V3CV6DV9EV12  
FV15G" RETURN
```

としてみましょ。だんだん音を大きくしながらドレミファと演奏していきます。

「V」の後に書ける数字は0～15までです。数が大きくなるほど音も大きくなります。「V0」を指定すると音を出しません。

電源を入れたときは「V8」の大きさです。
ゲームなどでは、V12～V14程度で使うと適正レベルで、プログラムが組めます。

へ) S、M…^{ねいろ か}音色を変える

「S」、「M」と^{すうじ}数字を^あ組み合わせて^{つか}使うと、^{ねいろ か}音色を変えることができます。たとえば^{つぎ}次のように^{うちこ}打ち込んでください。

PLAY "S10M3000CDEFGAB"

RETURN

「S」、「M」の^{あと}後の^{すうじ}数字をいろいろ^か変えてみましょう。
^{さまざま}様々な^{ねいろ}音色を^だ出すことができます。Sは0～15、Mは
1～65535の^{あいだ}間の^{すうじ}数字で^{しってい}指定することができます。

H 1 は、^{じゅうわ おん}2重和音、^{じゅうわ おん}3重和音も^だ出せます。

より^{くわ}詳しい^{せつめい}説明は、ベーシック^{ふんぽう へん}文法編 228 ページを
ご^{らん}覧ください。

おな 同じような処理にはサブルーチンを使おう! ゴ-サブ リターン つか かた …GOSUB～RETURNの使い方

プログラムを作っていると同じ処理があちこちにてくることがよくあります。こんなとき、同じ文を2度も3度も書くのは大変手数のかかるものです。そんなとき、プログラムのある部分が共通に何度でも使えると便利です。

プログラムの共通した部分を独立したプログラムにすることができます。このようなプログラムをサブルーチンプログラムと言います。サブルーチンを使うための命令がGOSUB～RETURNなのです。

プログラム

ある処理を
したい
↓
同じ処理を
もう1度したい
↓
END

あんなに同じ処理を2度も書くのはめんどくさいなァ...



● GOSUB～RETURN (ゴ-サブ～リターン)

GOSUBはサブルーチンに飛んで行きなさいという命令です。

GOSUB ぎょうばんごう
行番号

とプログラムの中に書いておくと「行番号」(普通、サブルーチンの先頭です)へ飛んで、そこで処理を行います。ここまではGOTO命令と同じですね。ただ違うのは、飛び先のサブルーチンで命令を実行していくうちにRETURN命令に出会うと、もとのGOSUB命令に戻ってきて次の命令を実行するという点です。

プログラム

GOSUB 100
↓
GOSUB 100
↓
END

100行からサブルーチンに
すれはいんだね!

サブルーチン

100
↓
RETURN



GOSUB~RETURN 命令を使って、計算練習のプログラムを作ってみましょう。RND、INT関数は説明しましたね。

```
10 K=0:CLS
20 FOR N=1 TO 10
30 X=INT(RND(1)*100)
40 Y=INT(RND(1)*100)
50 Z=X+Y
60 PRINT X;"+";Y
70 INPUT "こたえは";W
80 K=K+1
90 IF Z=W THEN GOSUB
   130 ELSE GOTO 60
100 NEXT N
110 PRINT K-10;"かい まちがいま
    した!"
120 END
130 PRINT"よく できました!"
140 PLAY"V13 L8CDFCDF CDL2F"
150 RETURN
```

行番号 130~150 がサブルーチンです。行番号90からサブルーチンに飛び込んでいるのがわかりますね。行番号 120の ENDを忘れずに入れてください。この ENDを忘れると、プログラムは行番号110から130、140、……と進んでしまい、行番号 150 でエラーになってしまいます。これは、GOSUB でサブルーチンに飛び込んだのではないのに、RETURN 命令を実行しようとしたためです。

59 + 10
こたえは?

RUN したあとの画面です。

〈起こりやすいエラー〉

●RETURN without GOSUB(リターン ウィズアウト ゴーサブ)

GOSUB 命令を実行していないのに、RETURN 命令に出会ったときに起こります。

はい ねつ なに 配列って何？

ディメンジョン
(DIM)

たとえば26個の変数に値を入力するプログラムを考えてみましょう。

```
10 INPUT AA
20 INPUT AB
30 INPUT AC
  ⋮
  ⋮
  ⋮
260 INPUT AZ
  ⋮
  ⋮
  ⋮
```

こんなふうを書いていたら大変です。変数を入力する部分だけで、26行ものプログラムになってしまいますね。もっと簡単に書く方法はないものではないでしょうか？
ベーシックには DIM 命令というものがあり、うまく使うとこのようなプログラムを非常に簡単に書くことができます。

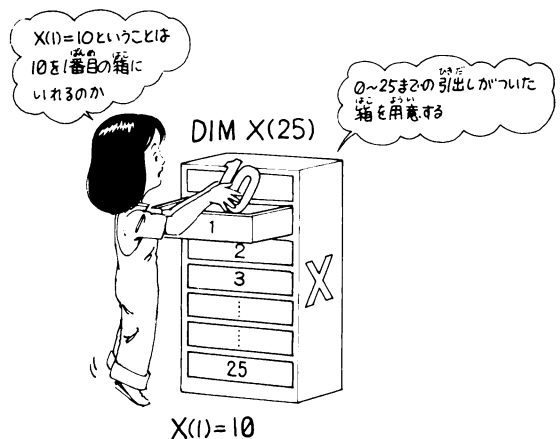
```
10 DIM A(25)
20 FOR N=0 TO 25
30 INPUT A(N)
40 NEXT N
  ⋮
  ⋮
  ⋮
```

26行あった部分が、わずか4行ですんでしまいました。

●DIM (ディメンジョン)

DIM X (25)

上の命令は「Xという名前の箱を用意しなさい。ただし、その箱には26の引出しがあって、0～25までの番号がついていますよ。」という意味です。



その引出しの中に数値をしまうことができます。たとえば、1 番目の引出しに10という数をしまいたいなら、

$X(1) = 10$

としてください。ほんとうに入っているかどうか確かめてみるには、

PRINT X(1)

とすればよいのです。

いまの説明をプログラムにしてRUNさせると次のようになります。

```
10 DIM X(25)
20 X(1)=10
30 PRINT X(1)
40 END
RUN
10
Ok
■
```

なんとなくわかってきましたか？

ここで使ったX()が配列と呼ばれるもので、引出しつきの変数なのです。

つまりDIM は、変数の引出しの数を宣言する命令で、これを配列の定義といいます。

添字といいます。

DIM 変数名(数値か変数か式)
 , 変数名() , ...

配列をたくさん使うときには、並べて書けます。

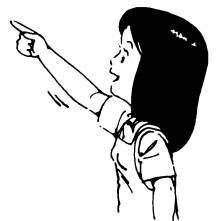
変数名のつけ方は、81 ページの「変数名の3つの規則」をよく読んでください。

配列を使うときは、変数名の後に番号をカッコではさんでつけます。(この番号を添字といいます) つまり

X という引出しの	0 番目	X(0)
"	1	" X(1)
⋮	⋮	⋮	⋮
"	25	" X(25)

$X(5) \times 6 + 1$ のように使う

カッコの中に数値、
 変数、式が書けるよ。
 $X(N)$ や $X(N+1)$ でよいんだ



かっこの中の番号は、DIM 命令で使ったかっこの中の数より大きくなってはいけません。また、プログラムの中では DIM 命令は配列を使うときより先に書かれていなければなりません。

次のプログラムは、20個の数字を入力して、その数の平均値を求めるものです。

```
10 PRINT "20コノ カズヲ イレテク  
    ダサイ"  
20 DIM D(20)  
30 S = 0  
40 FOR J = 1 TO 20  
50 PRINT J;: INPUT  
    "バンメ"; D(J)  
60 S = S + D(J)  
70 NEXT J  
80 A = S / 20  
90 PRINT "ヘイキン ハ"; A; "デス"  
100 END
```

行番号20で、0～20の21個の引出しをもつ変数Dを使えるようにし、行番号40で、FOR～NEXT命令でくり返しながら、Dに1つずつ数値を入れていきます。RUNさせてみましょう。

```
RUN  
20コノカズヲ イレテクダサイ  
1 バンメ? 10  
2 バンメ? 20  
3 バンメ? 15  
:  
20バンメ ? 101  
  
ヘイキン ハ 32.5 デス  
Ok  
■
```

●複雑な配列にチャレンジ

今までに説明した配列は引出しが縦1列だけでした。

H1は、縦、横に何列もの引出しが並んだ配列を使うこともできます。

たとえば、

```
10 DIM Y(5, 10)
```

と打ち込んでみましょう。この命令は「Yという名前
の箱を用意しなさい。ただしその箱には、縦に6列、
横に11列の引出しがあって、それぞれの0～5、0～
10までの番号がついていますよ」という意味です。

縦横2つの方向に引出しをもっているので、このよう
な配列を2次元配列と言います。まえで説明した縦方
向にしか引出しをもたない配列を1次元配列と言いま
す。

2次元配列を使うときには、

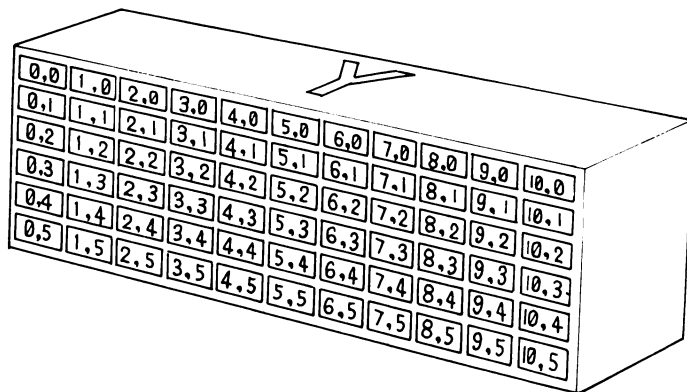
変数名（縦の引き出しの位置、横の引き
出しの位置）

として、かっこの中に2つの数値を指定する必要があ
ります。たとえば、

```
10 DIM Y(5, 10)
20 Y(1, 1)=15: Y(2, 5)=17
30 PRINT Y(1, 1), Y(2, 5)
40 END
```

のように使うわけです。

```
10 DIM Y(5, 10)
20 Y(1, 1)=15: Y(2, 5)=17
30 PRINT Y(1, 1), Y(2, 5)
40 END
RUN
    15                17
Ok
```



次のプログラムは、3人の英語、数学、国語のテスト
の点数を入力するときれいに並べて表示します。

```

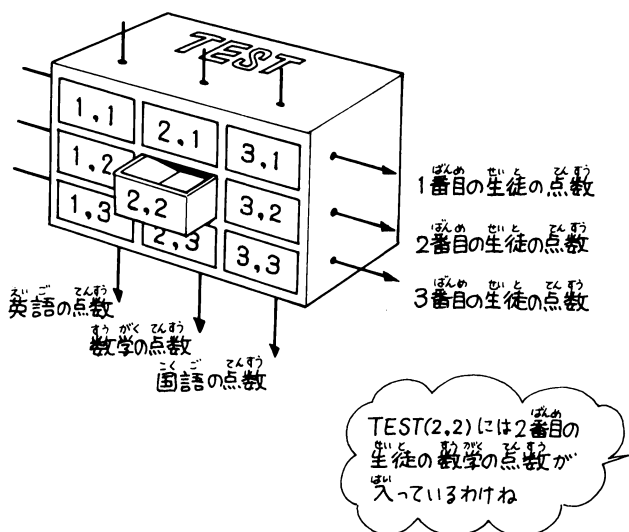
10 DIM TEST (3, 3)
20 PRINT "えいご, すうがく, こく
   ごの じゅんばんで いれること!"
30 FOR K=1 TO 3
40 PRINT "NO";K;:INPUT "の と
   くてんは";TEST (1, K), TEST
   (2, K), TEST (3, K)

```

```

50 NEXT K
60 CLS
70 PRINT SPC(6); "えいご" ; SPC
   (3); "すうがく"; SPC(4); "こく
   ご"
80 FOR K=1 TO 3
90 FOR I=1 TO 3
100 LOCATE 0, 3 * K : PRINT K
110 LOCATE (I * 8-3), 3 * K
   : PRINT TEST (I, K)
120 NEXT I
130 NEXT K
140 END

```



RUN させたらひとりずつ、英語、数学、国語の点数を
「,」でくぎって入れてください。

RUN

えいご、すうがく、こくごの しゅんはんで
いれること！

NO 1 の とくてんは？ 50,45,58

NO 2 の とくてんは？ 85,79,92

NO 3 の とくてんは？ 70,60,72

	えいご	すうがく	こくご
1	50	45	58
2	85	79	92
3	70	60	72

Ok

■

●文字の配列もあるよ

数値変数と同じように、文字変数にも配列を作ることができます。

```
10 DIM S$(2)
```

としてみましょう。これでS\$(0)～S\$(2)までの
文字の変数が3つとれたわけです。次に

```
20 S$(0)="わたしは ":S$(1)="MB-H1":S$(2)="です"
```

```
30 FOR N=0 TO 2:PRINT S$(N);:NEXT N
```

```
40 END
```

としてRUNさせてみましょう。ちゃんと配列に文字
が入っていますね。

```
10 DIM S$(3)
20 S$(0)="わたしは ":S$(1)="MB-H1 "
   S$(2)="です"
30 FOR N=0 TO 2:PRINT S$(N);
   :NEXT N
40 END
RUN
わたしは MB-H1 です
Ok
■
```

文字変数でも、配列をすることができます。

〈起こりやすいエラー〉

●Redimensioned array(リディメンジョン ド アレイ)

すでに宣言した配列をもう一度宣言した場合に起こり
ます。たとえば、

```
DIM A(20)
```

という命令を2回実行した場合などです。

●Subscript out range(サブスクリプト ア ウト オブレンジ)

配列の添字が、宣言した範囲をこえている場合です。

たとえば、

```
DIM A(20)
```

と配列を定義したのに

```
A(21)
```

などを使った場合です。

リード データ つか READ～DATAでデータを使おう

リード データ リストア
(READ, DATA, RESTORE)

変数に値を入れるときには、A=0やB=10のような
代入文のほかに、INPUT 命令を使ったりしました。
実はもう一つ方法があります。それが READ～
DATA 命令です。

READ X,Y,Z……いくつかの変数
DATA 10,20,30……それに対する数値

● READ～DATA (リード～データ)

次のプログラムを RUN してみましょう。

```
10 READ X, Y, Z
20 PRINT X + Y + Z
30 DATA 10, 20, 30
40 END
```

```
RUN
△0
□k
■
```

行番号10の READ X,Y,Z は「X,Y,Z の値
を読み込みなさい」という意味です。読み込まれる
X,Y,Zの値はどれかという行番号30の DATA 命
令後の10と20と30です。

READ X:READ Y,Z
DATA 10:DATA 20:DATA 30
などと READ～DATAを2つや3つに
切って使ってもいいよ



READ
DATA



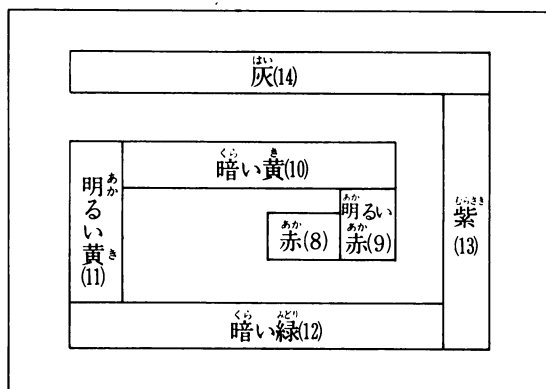
このようにして READ~DATA 命令を使って変数に数値や文字を入れることができます。決まっている値を変数に入れる場合にとても便利な命令です。INPUT 命令は、RUN させるたびに変数の値を入力しなければならないので変数の値が決まっている場合は、かえて不便なのです。

READ~DATA 命令と、前に学んだ LINE を使って、ちょっとした図形を描いてみましょう。

```

10 SCREEN 2
20 READ X0 , Y0, X1, Y1
30 LINE (X0, Y0)-(X1, Y1), 8,
   BF
40 FOR I=9 TO 14
50 READ X, Y
60 LINE -(X, Y), I, BF
70 NEXT I
80 GOTO 80
90 END
100 DATA 100, 100, 125, 120
110 DATA 150, 80, 75, 60,
      55, 140, 180, 160,
      200, 40, 55, 20

```



このプログラムでは、DATA より前の行にEND がありますが、DATA は、プログラムのどの部分に書いてもよいのです。

また、READ~DATA 命令で、文字列を文字変数に読み込めますこともできます。次のプログラムを打ち込んで、RUN させてみましょう。

```

10 CLS
20 READ X $
30 IF X $ = "%" THEN PRINT :
   GOTO 20
40 IF X $ = "$" THEN END
50 PRINT X $ ;
60 GOTO 20
70 END
80 DATA ムカシ ムカシ , アルト
   コロニ , オジイサント ,
   オバアサンガ ,
90 DATA スンティマシタ , %, オジ
   イサンハ ヤマヘ シバカ
   リニ ,
100 DATA オバアサンハ , カワヘ ,
   センタクニ , イキマシタ。
   , $

```

RUN

ムカシ ムカシ アルトコロニ オジイサント オバアサン
 ガ スンティマシタ
 オジイサンハ ヤマヘ シバカリニ オバアサンハ カ
 ワヘ センタクニ イキマシタ。

Ok



● RESTORE (リストア)

READ~DATA 命令によって読み込まれるデータは、一度読まれるともう二度と読み込まれることはありません。でも、同じデータを何度も使いたいということはよくあることです。

そのために、一度読み終ったデータをもう一度読み込む命令があります。それが RESTORE 命令です。

RESTORE 行番号

とすると、次からは、「行番号」の場所にあるデータから読み込みを始めます。

```
10 READ A, B, C
20 RESTORE 70
30 READ A$, B$
40 PRINT A + B + C ; A$ + B$
50 END
60 DATA 121
70 DATA 202, 57
```

上のプログラムは、まず行番号10の READ 命令で、行番号60, 70にある121、202、57の数を読みます。次に RESTORE 命令で行番号70からを指定しています。従って行番号30の READ 命令でA \$には「202」、B \$には「57」がはいることになります。 さっそく RUN させてみましょう。

```
RUN
380 20257
Ok
■
```

行番号20の RESTORE 70 を取ってしまったらこのプログラムはどうなるでしょうか。

```
20 RETURN
RUN RETURN
```

としてみましょう。行番号30の READ 命令では、もう読み込むべきデータが残っていません。READ~DATA 命令で、読み込むデータの数がないとエラーになってしまいます。

```
20
RUN
Out of DATA in 30
Ok
■
```

READ A\$, B\$, C\$...いくつかの文字変数
↑ ↓ ↘
DATA わたしは、MB-H1, です...対応する文字
 () " " 「 」で
 かこまなくてもよい。

〈起こりやすいエラー〉

● Out of DATA(アウト オブ データ)

READ~DATA 命令で、データの数がない場合に起こります。314 ページをお読みください。

とっても便利なON~GOTO, ON~GOSUB

GOTO 命令や、GOSUB 命令はもう前に勉強しましたね。ここでは、これらの機能をもう少し拡大した ON~GOTO、ON~GOSUB 命令をためしてみしましょう。

たとえば、「あなたのすきなくだものは?」という質問に対して「1.りんご 2.みかん 3.バナナ 4.メロン 5.パイナップル」というように、いくつかの中から選ぶようなとき、今までに勉強した命令を使うとすれば、IF ~THENを次のように使うでしょう。

```
10 INPUT "あなたの すきな くだものは? 1.りんご 2.みかん 3.バナナ 4.メロン 5.パイナップル"; X
20 IF X=1 THEN GOSUB 100
30 IF X=2 THEN GOSUB 200
40 IF X=3 THEN GOSUB 300
50 IF X=4 THEN GOSUB 400
60 IF X=5 THEN GOSUB 500
70 IF X<1 OR X>5 THEN 10
80 END
100 PRINT "りんご"
110 RETURN
200 PRINT "みかん"
210 RETURN
300 PRINT "バナナ"
310 RETURN
400 PRINT "メロン"
410 RETURN
```

```
500 PRINT "パイナップル"
510 RETURN
```

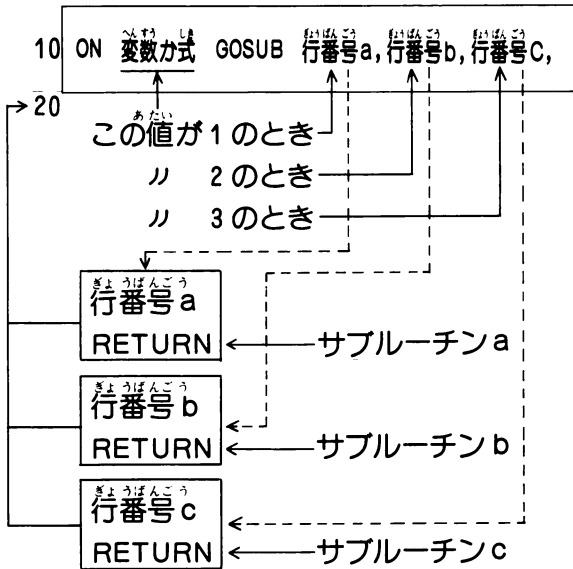
こんなときに便利なのが ON GOTO、ON GOSUB です。前のプログラムに ON GOSUB を使うと

```
10 INPUT "あなたの すきな くだものは? 1.りんご 2.みかん 3.バナナ 4.メロン 5.パイナップル"; X
20 ON X GOSUB 100, 200, 300, 400, 500
70 IF X<1 OR X>5 THEN 10
80 END
100 PRINT "りんご"
110 RETURN
200 PRINT "みかん"
210 RETURN
300 PRINT "バナナ"
310 RETURN
400 PRINT "メロン"
410 RETURN
500 PRINT "パイナップル"
510 RETURN
```

前のプログラムの20行目から、60行目までの5行が1行にまとまりましたね。

● ON ~ GOSUB (オン～ゴースブ)

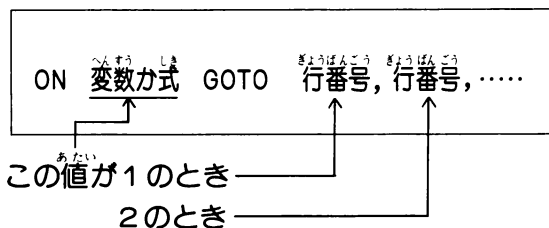
ON の後に書かれた変数の値によって、飛び先のサブルーチンを指定できる命令です。



RETURN 命令によって ON ~ GOSUB 命令の次の命令に戻ることに注意してください。また ON の後に書かれた変数、または式の値が0か、GOSUB の後に書かれた行番号の個数よりも大きいときは、ON ~ GOSUB 命令は無視されて次の行が実行されます。この値が負の場合は、エラーになるので気をつけてください。

● ON ~ GOTO (オン～ゴートウ)

ON の後に書かれた変数の値によって、プログラムの飛び先を指定できる命令です。



たとえば、ON の後の変数または式の値が2のときには GOTO の後にある2番目の行番号に飛ぶことになります。

また ON の後に書かれた変数や式の値が0や負になったり、GOTO の後に書かれた行番号の個数よりも大きいときには、ON ~ GOSUB と同じように無視されたり、エラーが出たりします。

```
10 INPUT X
20 ON X GOTO 40, 50, 60
30 A $ = "X=0 OR X>3" : GOTO 70
40 A $ = "X=1" : GOTO 70
50 A $ = "X=2" : GOTO 70
60 A $ = "X=3"
70 PRINT A $
80 GOTO 10
90 END
```

上のプログラムを打ち込んで RUN させてみましょう。キーボードからいろいろな数を入力してください。どんな結果が出ましたか？

```
RUN
? 3
X=3
? 1.2
X=1
? 0
X=0 OR X>3
? 5
X=0 OR X>3
? -10
Illegal function call in 20
Ok
■
```

〈起こりやすいエラー〉

- Undefined line number (アンディファインド ライン ナンバー)
GOTO, GOSUB 命令の飛び先がない場合です。
- Illegal function call (イリーガル ファンク ション コール)
ON の後の変数または式の値が負になった場合に起こります。

関数を作ってみよう

デファイン ファンクション
(DEF FN)

今までに説明した関数は、すでにベーシックに用意されていて、内容が決められている関数でした。

ここでは、自分だけの関数を新しく作る方法について説明します。そのときに使うのがDEF FNです。

● DEF FN (デファイン ファンクション)

絶対値を計算するのはABS という名前の関数でした。このように、関数には処理の内容(たとえば絶対値を計算する)と関数の名前(たとえばABS)を持っています。

DEF FN を使った関数定義では、処理の内容は好きなものを選べます。ただし、関数の名前に多少のきまりがあります。「先頭の2文字はFNでなければならない。」ということです。

DEF FN 関数の名前 (変数, 変数, …) = 式

- ・関数の名前の先頭に必ずFNをつけること。
- ・名前の規則は変数名と同じ。(⇒81ページ)

さて、それでは、円の面積を計算する($3.14159 * X^2$)「FNEN」という関数を作ってみましょう。「FNEN」が関数の名前ですが、FNは決まっていますので、その後にENとつけただけです。

DEF FNEN (X) = 3.14159 * X ^ 2

このように書けばよいのです。ただしDEF FN 文はプログラム中でしか使えません。

次のようなプログラムを作ってみましょう。

```
10 DEF FNEN (X) = 3.14159 * X ^ 2
20 INPUT "X = "; X
30 PRINT FNEN (X)
40 GOTO 20
50 END
```

行番号30で PRINT FNEN (X) とするたびに
3.14159 * X²が計算されているのがわかりますね。
こんどは文字列を使う関数を定義してみましょう。
関数の名前の最後に「\$」をつけるだけで、あとは数
値関数の場合と同じです。

```
DEF FN関数の名前$ (変数, 変数, ...) =式
```

- ・ 名前の最後に「\$」をつけると文字列をかえす関数

次のようにプログラムしてRUN してみよう。

```
10 DEF FNM$(X$, M, N)=  
    MID$(X$, LEN(X$)-M-N  
    +2, N)  
20 INPUT "X$, M, N=" ; X$,  
    M, N  
30 PRINT FNM$(X$, M, N)  
40 GOTO 20  
50 END
```

この FNM\$(X\$, M, N) という関数はX\$の右
からM番目の文字から左へ向ってN個の文字を左から
ひろい出すように定義したものです。ちょうど MID\$
と逆の機能ですね。

```
RUN  
X$,M,N=? 1234567890,1,5  
67890  
X$,M,N=? 1234567890,4,3  
567  
X$,M,N=?  
Break in 20  
Ok  
■
```

〈起こりやすいエラー〉

● Undefined user function(アンディファ インド ユーザ ファンクション)

DEF FN 命令で定義していないユーザ関数を使
おうとした場合に起こります。

また、ユーザ関数をプログラム中で定義する前に使
った場合にも起こります。

- DEF FN は、プログラム中では、必ずその関数
を使う前に書かれていなければなりません。

H1の中身をのぞいてみよう

(PEEK関数, POKE, CLEAR)

さあ、ベーシックの長い道のりもあとわずかです。ベーシックというのはどんなものなのか、理解できたでしょうか？ スプライトを使って絵を動かしたり、PLAY 文で音楽を演奏したり、H1でなかなか楽しいことができるでしょう。

ここでは、ちょっとむずかしいかもしれませんがH1の中身をのぞいてみるお話をします。H1は、こんなこともできるのかという程度で読んでみてください。

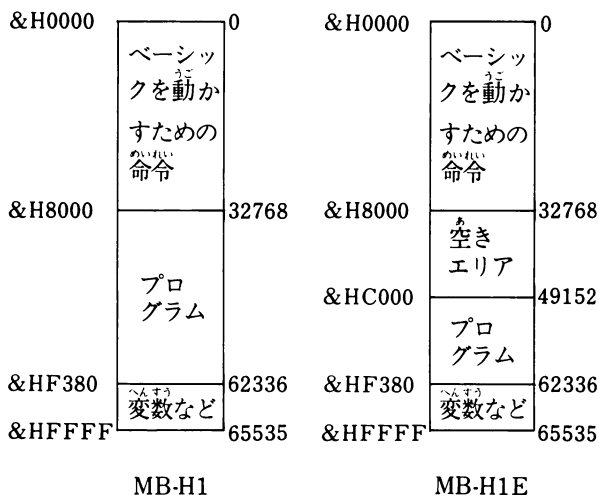
●メモリって何？

キーボードから文字や数字を打ち込んでも、画面がいつぱいになれば、どんどん消されてしまいました。しかし、先頭に行番号をつけてプログラムにしていれば、それをいつまでも（NEW とするか電源を切るまでは）覚えているのでしたね。

いったいH1は、プログラムをどこに覚えておくのでしょうか。じつは、H1にはベーシックで書かれたプログラムを覚えておいたり、ベーシックを動かすためのいろいろな命令をしまっておくための場所があります。それをメモリ（記憶装置）といいます。

メモリは使いやすように1つずつ0から順番に番号がふってあります。H1はメモリの0から65535（&HFFFF）まで使うことができるのです。このメモリにふった番号を番地と呼んでいます。

H1はメモリをどんなふうに使っているのでしょうか？ 番地ごとに、ちょっと説明してみましょう。



イ) 0～32767(&H7FFF)番地

ベーシックを動かすための命令をしまっておく場所です。ここで、プログラム中にあるさまざまな命令を理解して、実行していくのです。この部分は変化しません。

ロ) MB-H1:32768(&H8000)～65535(&HFFFF)番地 MB-H1E:49152(&HC000)～65535(&HFFFF)番地

プログラムや変数をしまっておくための場所です。プログラムの作成や、実行にともなって、この部分のメモリの内容は変化します。

メモリの使われ方はだいたい理解できましたか？ では実際にメモリに何が書いてあるのか、のぞいてみることにしましょう。

●PEEK (ピーク) 関数

メモリの中をのぞくときに使うのがこの PEEK 関数です。

PEEK(番地)

とすると、その番地のメモリの内容を値とするのです。さっそく使ってみましょう。

PRINT PEEK (0) RETURN

としてみましよう。

```
Ok
PRINT PEEK(0)
243
Ok
■
```

どんな値が画面に書かれましたか？ 243という数字が、メモリの0番地にはっていますね。かつこの中の数字をいろいろに換えて、もっと他の番地ものぞいてみましょう。

えっ？ なにに、数字しか出てこないのおもしろくないって？ それは当然なのです。0番地から32768番地まではベーシックを動かすための命令がはっているのですが、そもそもコンピュータは命令を数字で表現して覚えているのです。数字がどんな命令に当たるのかを知りたい人は機械語の勉強をすることをおすすめします。

●POKE, CLEAR (ポーク、クリア)

今度はメモリに好きな数字を書き込んでみましょう。そんなときに使うのが POKE 命令です。

POKE 番地, データ

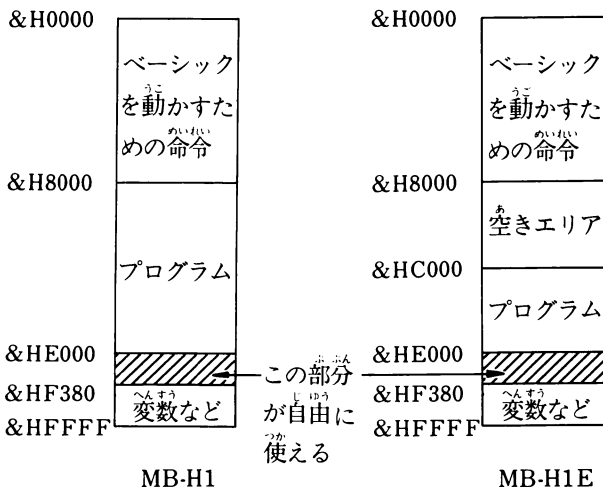
とすれば、その番地のメモリにデータを書き込めます。この命令を使う前に、もう一度さきほどの図を見てください。H1のメモリにかつてに変な値を書き込んでまいじょうぶなんでしょうか？ ちょっと心配ですね。メモリの中に、かつてに使ってもかまわない場所をとることはできないのでしょうか？

そんな時には CLEAR 命令を使います。CLEAR 命令はベーシックが使うメモリの上限を決める命令です。たとえば

CLEAR 300, &HFFFF

とすると、ベーシックは &HFFFF 番地までしかメモリを使用せず &HE000 ~ &HF37F の4992個分が、自由に使えるようになるわけです。

CLEAR 文字領域の大きさ, 番地



CLEAR 300, &HFFFF 実行後

さて、それではPOKE ^{めいれい}命令でメモリに何か書き込^にんでみましょう。

POKE &HE001,100 RETURN

としてください。本当^{ほんとう}に書き込^にむことができたでしょうか？ PEEK ^{かんすう}関数を使^{つか}って確^{たしか}めてみましょう。

PRINT PEEK(&HE001) RETURN

だいじょうぶ、ちゃんと書かれていましたね。

```
Ok
POKE &HE001,100
Ok
PRINT PEEK(&HE001)
100
Ok
■
```

PEEK ^{かんすう}関数やPOKE ^{めいれい}命令を使^{つか}うと、H1の世^せ界はさらに広^{ひろ}がりますが、そのためには機^き械^{かい}語^ごの知^ち識^{しき}が必^{ひつ}要^{よう}です。機^き会^{かい}があればぜひ機^き械^{かい}語^ごも勉^{べん}強^{きやう}してください。

〈起^おこりやすいエラー〉

●Illegal function call(イリーガル ファンク ションコール)

POKE ^{めいれい}命令で書き込^にむデータが0~255の範^{はん}圍^いにない場^{ばい}合^あに起^おこります。たとえば、

POKE &HD200,500

などとした場^{ばい}合^あです。

●Overflow(オーバーフロー)

PEEK,POKE,CLEAR で指^し定^{てい}する番^{ばん}地^ちが0~65535の範^{はん}圍^いになかった場^{ばい}合^あに起^おこります。たとえば、

PEEK (65536)

などとした場^{ばい}合^あです。

●Out of memory(アウト オブ メモリー)

CLEAR ^{めいれい}命令で、ベーシックで使^{つか}うメモ^{メモ}リ^リの^{しうげん}上^{ちい}限^{げん}を小^こさくとりすぎた場^{ばい}合^あです。も^もっ^もと^と大^{おお}き^きな^{あたい}値^ちにするか、プ^{しや}ロ^ろグ^くラ^らム^むを短^{たん}く^くするかしてくだ^{くだ}さい。

ベーシック文法編^{ぶんぽう へん}

「ベーシック基礎編^{きそ へん}」を理解^{り かい}して、自分でいろいろなプログラム^{ぷろぐらむ}を作り^{つく}はじめると、もっと楽しく、もっといろいろなことにH1^{えいち 1}が使える^{つか}そうだと感じる^{かん}ことがある^{おも}と思います。そんなときに、便利な^{べん り}コマンドやステートメント、関数^{かん すう}を知^しっていると、意外^{い がい}に簡単な^{かんたん}方法^{ほう ほう}で、H1^{えいち 1}に思^{おも}ったとおりのことをさせる^{つか}ことができたりします。

ベーシック文法編^{ぶんぽう へん}では、H1^{えいち 1}で使う^{つか}ことのできる命令^{めい れい}や、関数^{かん すう}を、機能別^{きののう べつ}に分^わけて1つずつ説明^{せつ めい}します。アルファベット順^{じ ほん}の索引^{さく いん}もありますので、プログラム作成^{さく せい}時の辞書^{じ しょ}がわりに使う^{つか}と便利^{べん り}です。

はじめに

ベーシック文法編では、H 1 で使用できるベーシックのコマンド ステートメント 関数をすべて説明します。

このベーシック文法編は、ベーシック基礎編と違い、全文を読んではいながらH 1 の動作を確認していくというよりも、皆さんが実際に目的を持ったプログラムを作成していくときに必要となる関数やコマンドなどを調べるのに便利になっています。つまり、ベーシック基礎編を教科書とすると、ベーシック文法編は辞書ということになります。

ベーシック文法編には、基礎編で説明していないコマンドやステートメント、関数が多く出てきますが、新しいものに積極的にチャレンジして、どんどん自分のものにしていきましょう。

ベーシック文法編は、機能別に分類して説明してありますが辞書として使用していただくのに便利のようにA B C 順（アルファベット順）の索引も、279ページにあります。

MSX-BASICの概要

1. 行の形式

n n n n ベーシックのステートメント[: ベーシックのステートメント……]

- n n n n は行番号です。
- 1 行には、ベーシックのステートメントを「:」(コロン)で結ぶことにより、複数のベーシックのステートメントを書くことができます。
- 1 行には、255文字まで書くことができます。

2. 行番号

- 0から65529までの整数を使用します。
- LIST、AUTO、DELETE命令で「.」(ピリオド)を使用すると、現在の行番号を参照します。

3. 使用可能な文字

- 英大文字、英小文字、ひらがな、カタカナ、数字(0~9)、英記号、かな記号、グラフィック文字が使用できます。それぞれの内容は、取扱編39ページから45ページをご覧ください。

4. 定数

- 数値定数と文字定数とがあります。
- 文字定数
最大255文字までの文字列で「"」(ダブルクォート)ではさんで使用します。
例. "ABCD"
"230B"
- 数値定数
数値の表現の形式で次の5種類にわけられます。
 - i) 固定小数点定数
通常用いている正負の実数です。
例. 127
-703
12.76

ii) 浮動小数点定数

- 指数形式で表わされる正負の数値です。
次の形式で表わされます。

$\pm n \cdots n . n \cdots n E \pm n n$

または

$\pm n \cdots n . n \cdots n D \pm n n$
↑ ↑
固定小数点定数部 指数部($10^{-64} \sim 10^{63}$)

例. 235.488 E -7 ($= 0.0000235488$)
-611 D +5 ($= -61100000$)

iii) 2進定数

- 2進法で表わされた数値です。
- 数値の前に「&B」をつけます。

例. &B10010 ($= 18$)

iv) 8進定数

- 8進法で表わされた数値です。
- 数値の前に「&O」をつけます。

例. &O777 ($= 511$)

v) 16進定数

- 16進法で表わされた数値です。
- 数値の前に「&H」をつけます。

例. &H7FFF ($= 32767$)

● 数値の型について

内部の表現形式で次の3種類に分けられます。

i) 整数型

- 数値の後に「%」(パーセント)をつけます。
- -32768~32767の範囲の整数です。

例. -3000%

27%

ii) 単精度型

- 数値の後に「!」(感嘆符)をつけます。
- 6桁まで記憶、表示されます。(7桁目を四捨五入します。)

。また「E」を使った指数形式もこの型になります。

例. -2.13E-13
26.5!

iii) 倍精度型

- ・数値の後に「#」(シャープ)をつけます。
- ・14桁まで記憶、表示されます。(15桁目を四捨五入します。)

。また、「D」を使った指数形式もこの型になります。

例. -611D+5

- 特に指定を行わなければ、倍精度で記憶、計算、表示を行います。

定数のまとめ

定数	表現形式	型	数値例
数値定数	固定小数点	整数型	12%, -32000%
		単精度型	1.25!, 57000!
		倍精度型	1.2, -256#
	浮動小数点	単精度型	1.0E+7
		倍精度型	-2.56D-21
		整数型	&B10011
		整数型	&O2777
		整数型	&HABFF
文字定数	文字列	文字型	"ABCD", "-23AF"

5. 変数

- 変数名の長さは、255文字まで可能ですが、3文字目以降の文字は識別されません。
- 変数名には、英文字、数値を使用しますが、先頭の1文字目は必ず英文字でなければなりません。
- ベーシックの命令、関数は変数名に使用できません。また、命令、関数を変数名の中に含んでもいけません。
 - ・詳細は基礎編81ページをご覧ください。
- 変数には、文字変数と数値変数があります。
- 文字変数には、変数名の後に「\$」(ドル記号)をつけ、255文字まで入れることができます。
- 数値変数にはつぎの3種類があります。
 - i) 整数型変数……変数名の後に「%」(パーセント)記号をつけます。
 - ii) 単精度型変数…変数名の後に「!」(感嘆符)記号をつけます。

- iii) 倍精度型変数…変数名の後に何もつけないか、または「#」(シャープ)をつけます。

- DEFINT, DEFSG, DEFDBL, DEFSTR を使用すると、それぞれ変数の型を宣言することができます。詳細は文法編211ページをご覧ください。
- 変数を記憶するメモリの大きさはつぎのとおりです。

整数型変数	2 バイト
単精度型変数	4 バイト
倍精度型変数	8 バイト
文字変数	入れる文字数 + 3 バイト

6. 型变换

- 数値定数を文字変数に、またはその逆に変換する場合には、関数STR\$, VALを用います。
- ある型の数値(定数または変数)を他の型の型に変換するには、その型の数値を、希望する型の数値変数に代入します。
- 式の計算は、演算数値中の一番高い精度に変換してから行なわれます。ただし、返される数値は、指定がある場合には指定された精度になります。

例.	10	D = 1 / 3 !	} <div style="display: inline-block; vertical-align: middle;"> えんざん ばいせい ど おに 演算は倍精度で行な われます。 結果は倍精度で表示 されます。 えんざん ばいせい ど おに 演算は倍精度で行な われます。 結果は単精度で表示 されます。 </div>
	20	PRINT D	
	RUN		
	.3333333333333333		
	10	D ! = 1 / 3	} <div style="display: inline-block; vertical-align: middle;"> えんざん ばいせい ど おに 演算は倍精度で行な われます。 結果は倍精度で表示 されます。 えんざん ばいせい ど おに 演算は倍精度で行な われます。 結果は単精度で表示 されます。 </div>
	20	PRINT D !	
	RUN		
	.3333333		

- 論理関係式の演算数は、整数に変換され、結果は整数で返されます。演算数は、-32768から、32767の範囲の数値で、それ以外は、Over flow エラーとなります。

- 浮動小数点(実)数値を整数に変換すると、小数部分は、切り捨てられます。

- 単精度型を倍精度型変数に変換する場合には、
上位 6 桁のみが有効となります。これは、単精度
型数値が 6 桁の精度しかもっていないからです

7. 式

- 文字^{もじ}または数値^{すうち}の定数^{ていすう}、変数^{へんすう}あるいはそれらの組合せ^{くみあわせ}を、演算子^{えんざんし}を用いて一つの値^ちを作りだすものを式^{しき}といいます。

- 式には、算術式、関係式、論理式、関数の4種類があります。

- さんじゅつえんさん し
算術演算子

つぎの演算子を用います。例

+	加 ^か 算 ^{さん}	$\times + Y$
-	減 ^{げん} 算 ^{さん}	$\times - Y$
*	乗 ^{じょう} 算 ^{さん}	$\times * Y$
/	除 ^{じょ} 算 ^{さん}	\times / Y

へ	へき乗 <small>じゆう</small>	$X \wedge Y$
－	ふ数 <small>ふすう</small>	$-X$
≡	整数除算 <small>せいすう じゆ ざん</small>	$X \equiv Y (10 \equiv 4 = 2)$
MOD	整数剰余 <small>せいすう じゆう じよ</small>	$X \text{ MOD } Y$ ($10.4 \text{ MOD } 4 = 2$)

- ## ●比較演算子

- i) 2つの^{あた}値の^ひ比較^{かく}に^{つか}使います。
- ii) つぎの^{えん}演算子^{ざん}を^し用い^{もち}ます。

例

=	^{ひと} 等しい	$X = Y$
>	^{ひと} 等しくない	$X > Y$
<	^{みまん} 未満	$X < Y$
>	^{ちゆう} 超	$X > Y$
<=	^い 以下	$X \leq Y$
>=	^{いじゆう} 以上	$X \geq Y$

- iii) 1つの式の中に、算術式^{しき けいしき}と関係式^{かんけいしき}の両方^{りやうほう}がある場合には、算術式が評価された後に、関係式^{かんけいしき}が評価^{ひやうか}されます。

- 論理演算子

- i) 論理演算子は以下の6種類があり、整数型の数値に適用できます。数値の各ビットについて以下の演算を行うことに注意してください。

えん ざん し 演算子	し よう ほう 使用法	ひょう じ 評価															
NOT (ノット)	NOT X	Xではない <table><tr><th>X</th><th>NOT X</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	X	NOT X	0	1	1	0									
X	NOT X																
0	1																
1	0																
AND (アンド)	X AND Y	Xであり、かつYである <table><tr><th>X</th><th>Y</th><th>X AND Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	X	Y	X AND Y	0	0	0	0	1	0	1	0	0	1	1	1
X	Y	X AND Y															
0	0	0															
0	1	0															
1	0	0															
1	1	1															
OR (オア)	X OR Y	Xまたは、Yである <table><tr><th>X</th><th>Y</th><th>X OR Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	X	Y	X OR Y	0	0	0	0	1	1	1	0	1	1	1	1
X	Y	X OR Y															
0	0	0															
0	1	1															
1	0	1															
1	1	1															

演算子	使用法	評価
XOR (イクスクルー シブ・オア)	X XOR Y	XとYは等しくない
		X Y X XOR Y
		0 0 0
		0 1 1
		1 0 1
EQV (イクワイバレント)	X EQV Y	XとYは等しい
		X Y X EQV Y
		0 0 1
		0 1 0
		1 0 0
IMP (インプライ)	X IMP Y	XはYを含む
		X Y X IMP Y
		0 0 1
		0 1 1
		1 0 0

例1. NOT -2 1

$$\begin{array}{l} -2 = \&B1111111111111110 \\ \text{NOT } -2 = \&B0000000000000001 = 1 \end{array}$$

例2. 23 AND 7 7

$$\begin{array}{l} 23 = \&B10111 \\ \text{AND } 7 = \&B00111 \\ \hline \&B00111 = 7 \end{array}$$

例3. 8 OR 6 14

$$\begin{array}{l} 8 = \&B1000 \\ \text{OR } 6 = \&B0110 \\ \hline \&B1110 = 14 \end{array}$$

例4. 16 XOR 6 22

$$\begin{array}{l} 16 = \&B10000 \\ \text{XOR } 6 = \&B00110 \\ \hline \&B10110 = 22 \end{array}$$

例5. 7 EQV 8 -16

$$\begin{array}{l} 7 = \&B0000000000000011 \\ \text{EQV } 8 = \&B0000000000001000 \\ \hline \&B11111111111110000 = -16 \end{array}$$

例6. 12 IMP 5 -9

$$\begin{array}{l} 12 = \&B0000000000001100 \\ \text{IMP } 5 = \&B0000000000000101 \\ \hline \&B11111111111110111 = -9 \end{array}$$

ii) 論理演算子は演算子の中で最も低い優先度となります。

●関数

i) 与えられた演算数値をもとに、あらかじめ決められた演算を行ない、その結果の値を返すものが関数です。

ii) プログラム中で、ユーザが設定した関数を使うこともできます。(DEF FN関数) 基礎編 176 ページを参照してください。

●文字列の演算

i) 文字列は「+」記号を使って、つなぐことができます。

ii) 数値の比較演算子を用いて、文字列の比較を行なうことができます。比較の結果が真ならば -1、偽ならば 0 となります。

iii) 文字列の比較演算は、文字列の先頭から一文字ずつキャラクターコードを比較して行なわれます。もし比較の途中でキャラクターコードが異なった場合は、小さいコードを含む文字列が小さいとされます。また一方の文字列が終りになった場合には文字列の短い方が、小さいとされます。

iv) 「」(空白)も比較の対象になります。

例. "MB"+"-"+"H1"="MB-H1"

"RED"<>"BLACK"

"RECORD"<"RECORDER"

"IN PUT">"IN PUT"

8. エラー

●プログラム実行中、エラーが発生した場合、画面にはエラーメッセージが表示されます。(付録 314 ページ)

●エラーが発生した時点でプログラムの実行は中断され、コマンド待ちの状態になります。

●中断されたままの状態では、変数の値、プログラムは残っています。

●プログラム中のエラーは、その行が実行されるまで検出されません。

9. 文法編の使い方

- 命令、ステートメント、関数は、それぞれ機能別（理解しやすい順序）になっています。辞書のようにして目的の命令語をさがすときは、279ページの索引をお使いください。
- 説明文中の書式について
 - i) アルファベットの`大文字`で示されている項目はそのまま入力します。
 - ii) カギカッコ`<>`で囲まれた項目は、あなたが指定します。
 - iii) 角カッコ`[]`で囲まれた項目は、省略することができます。
 - iv) 省略記号`……`の続く項目は、1行の許す範囲内で任意の回数を繰り返すことができます。
 - v)

;
または
,

のように書かれたものは、上下どちらを使っても、よいことを意味します。
 - vi) 上記以外の記号は、文法的に必要な記号ですから、プログラム中に書かれなければなりません。
 - vii) 解説などで出てくる「&H」に続く数値は16進数であることを示します。
 - viii) 「-」は、マイナス記号を示します。

コマンド

LIST (リスト)

➡ 基礎編 84 ページ

機能 プログラムリストを画面に出力します。

書式 LIST [<始点行番号>] [- [<終点行番号>]]

文例 LIST 100-300
(行番号100から300までのプログラムを画面に表示します。)

解説

現在メモリ上にあるベーシックプログラムの一部、または全部を画面に表示します。2つの行番号を省略するとプログラム全部を、<始点行番号>を省略するとプログラムの最初から<終点行番号>まで、

<終点行番号>を省略すると<始点行番号>以降プログラムの最後まで出力します。LIST コマンドの後に「.」(ピリオド)だけを入力すると、現在実行が停止している行を表示します。

LLIST

(エル リスト)

→ 基礎編 91 ページ

機能 プログラムリストをプリンタに出力します。

書式 LLIST [<始点行番号>] [- [<終点行番号>]]

文例 LLIST 100-300
(行番号100から300までのプログラムをプリンタに出力します。)

解説

現在メモリ上にあるベーシックプログラムの一部、行番号の指定はLISTの場合と同じです。
または全部をプリンタに出力します。リストする

AUTO

(オート)

→ 基礎編 93 ページ

機能 行番号を自動的に発生させます。

書式 AUTO [<行番号>][, <増分>]

文例 AUTO 100, 5
(行番号を100、105、110…の順に発生させます。)
AUTO , 5 とすると、行番号を0、5、10…の順に発生させます。

解説

このコマンドを実行すると、**[RETURN]** キーを入力するたびに、<行番号>に始まり<増分>ずつ増加していく行番号を自動的に発生させます。AUTO コマンドの後に「.」(ピリオド)を入力すると、現在実行が停止している行から、行番号を発生します。

注) すでに使用されている行番号を、AUTOで発生させた場合には、行番号のすぐ後に「*」が表示されて、プログラムが変更されることを警告します。このとき、**[RETURN]** だけを押しとその行は保護されます。

RENUM

(リナンバー)

➡ 基礎編 94 ページ

機能 プログラムの行番号をつけ直します。

書式 RENUM [<新行番号>] [, <旧行番号>] [, <増分>]

文例 RENUM 200, 100, 20
(行番号100を、200とつけなおし、以後20番飛びに行番号をつけかえます。)

解説

<旧行番号>で指定する行より後ろの行番号を、<新行番号>から始まり<増分>ずつ増えていく新しい行番号につけ直します。これらを省略した場合は次のとおり設定されます。

<新行番号> 10

<旧行番号> 現在あるプログラムの最も小さい行番号

<増分> 10

また、この命令で、GOTO、GOSUB、IF～THEN、

ON～GOTO、ON～GOSUB、ERLの中にある行番号もつけ直します。ただしこれらの行番号が、プログラムの中に存在していない場合、「Undefined line number」とエラー表示して、その行番号はそのまま残ります。
行番号の順序が変化するようなパラメータを与えると、Illegal function call エラーが出ます。

DELETE

(デリート)

➡ 基礎編 95 ページ

機能 指定した行を消します。

書式 ① DELETE <始点行番号> [<—終点行番号>]
② DELETE —<終点行番号>

文例 DELETE 50—150
(行番号50から150までを消します。)

DELETE —300
(最初の行から、行番号300までを消します。)

解説

〈始点行番号〉から〈終点行番号〉までのプログラムを消します。〈始点行番号〉のみを指定した場合はその行1行のみを消します。書式②の場合はプログラムの最初からその行までを消します。DELETE コマンドの後に「.」(ピリオド)だけを

入力すると、現在実行が停止している行を消します。〈始点行番号〉がプログラム中に存在しない場合は、次の行番号から実行します。また〈終点行番号〉がない場合には Illegal function call エラーが出ます。

NEW (ニュー)

→ 基礎編 82 ページ

機能 プログラムを消し、変数をクリアします。

書式 NEW

解説

メモリ上のベーシックプログラムを消し、すべての数値変数を0に、文字変数を「 」(ヌルストリング)にします。プログラム中にこの命令が現われ

ると、実行をやめプログラムを消し、変数をクリアして、命令待ちの状態になります。開かれているファイルは、そのままで閉じられません。

RUN (ラン)

→ 基礎編 83 ページ

機能 プログラムを指定された行から実行します。

書式 RUN [〈行番号〉]

文例 RUN 100
(行番号100の命令から実行します。)

解説

プログラムを〈行番号〉から実行します。〈行番号〉を省略した場合、プログラムの先頭から実行を始めます。

プログラムの実行に先だって、開かれているファイルを閉じます。

CONT

(コンティニュー)

➡ 基礎編 97 ページ

機能 中断したプログラムの実行を再開します。

書式 CONT

解説

[CTRL] + [STOP] 入力後、または STOP 文、END 文を実行後、コマンド待ちの状態です。CONT を入力すると、停止した次のステートメントから実行を再開します。INPUT 文で停止した場合は、その

INPUT 文から実行を再開します。ただし、停止後、プログラムの編集(追加、訂正、削除)を行なった場合、CONT 命令によって実行を再開することはできません。

TRON/TROFF

(トレース オン/トレースオフ)

機能 トレースモードの設定・解除を行ないません。

書式 TRON または TROFF

解説

① TRON

TRON 命令が実行されると、ベシックはトレースモードに入ります。このモードでは、実行したプログラムの行番号を「[」、「]」でくくって画面に表示します。おもにプログラムのデバッグに使用します。

② TROFF

トレースモードを解除します。また、NEW 命令を実行しても同じ結果が得られます。

サンプルプログラム

```
10 TRON
20 FOR N=1 TO 10
30 PRINT N
40 NEXT N
50 TROFF
60 END
```

(行番号20から50までのトレースを行ないません。)

CSAVE

(シー セーブ)

➡ 基礎編 87 ページ

機能 カセットテープにBASICのプログラムを記録します。

書式 CSAVE "<ファイル名>" [, <ボーレイト>]

文例 CSAVE "HINT", 2

(プログラムを "HINT" というファイル名で、カセットテープに2400ボーで記録します。)

解説

ベーシックエリアにあるプログラムに<ファイル名>をつけてカセットテープにセーブします。
<ボーレイト>はカセットレコーダへの書き込み速度を指定します。1、2の2種類が指定できます。

1 : 1200ボー

2 : 2400ボー

<ボーレイト>が省略された場合、SCREEN命令で指定された速度で書き込みを行ないます。
プログラムは短縮された中間言語の形で記録されます。

CLOAD

(シー ロード)

➡ 基礎編 89 ページ

機能 カセットテープからメモリへプログラムをロードします。

書式 CLOAD ["<ファイル名>"]

文例 CLOAD "HINT"

(カセットテープから "HINT" というファイル名のプログラムを読み込みます。)

解説

カセットテープから<ファイル名>で指定したプログラムを読み込みます。<ファイル名>を省略すると、

テープ上で最初に見つけたプログラムをロードします。ボーレイトは自動的に決定されます。

CLOAD?

(シー ロード ベリファイ)

➡ 基礎編 89ページ

機能 カセットテープにセーブしたプログラムの^{ないよう}内容とメモリにあるプログラムの^{ないよう}内容を^ひ比較^{かく}します。

書式 CLOAD? ["<ファイル名>"]

文例 CLOAD? "HINT"

(カセットテープにセーブした" HINT" というファイル^{めい}名のプログラムが、メモリ^{ない}内のプログラムの^{ないよう}内容と^{おな}同じかどうか、^{かくにん}確認します。)

解説

カセットテープのプログラムの^{ないよう}内容とメモリにあるプログラムの^{ないよう}内容が^{おな}同じであれば Ok と^{ひょうじ}表示し、

もしそうでなければ Verify error とエラー^{ひょうじ}表示されます。

LOAD

(ロード)

機能 プログラムをファイルから^よ読み^み込みます。

書式 LOAD "<デバイスディスクリプタ> [<ファイル名>]" [, R]

文例 LOAD "CAS: SAMPLE", R

(カセットから" SAMPLE" というファイル^{めい}名のプログラムを^よ読み^み込み、ただちに^{じっこう}実行します。)

解説

<デバイスディスクリプタ>で^し指定^{てい}した装置^{そうち}から、<ファイル名>のプログラムファイルをメモリに^よ読み^み込みます。(デバイスディスクリプタについては OPEN 命令を^{めい}参照^{さんしょう}してください。)

この^{めい}命令^{じっこう}を実行すると、すべてのファイルは^と閉じられ、すでにメモリに^{そんざい}存在していたプログラムは

^け消されます。ただしあとに R をつけた場合、ファイルは^と閉じられることなく、またプログラムを^よ読み^み込んだ後、ただちにそのプログラムを^{じっこう}実行します。

もし、<ファイル名>を省略すると、^{さいしよ}最初^{あはれ}に現れたプログラムを^よ読み^み込みます。

SAVE

(セーブ)

機能 メモリ内にあるベーシックのプログラムを指定した装置に記録します。

書式 SAVE "<デバイスディスクリプタ> [<ファイル名>]"

文例 SAVE "CAS: SAMPLE"

(カセットテープレコーダに「SAMPLE」という名前のプログラムを記録します。)

解説

<デバイスディスクリプタ>で指定した装置に、<ファイル名>をつけて、ベーシックのプログラムを記録します。

CTRL + **Z** (&H1A) が、ファイルの終了として記入されます。

プログラムはアスキー形式 (文字形式) で記録されます。

BLOAD

(ビー ロード)

機能 機械語プログラムをファイルから読み込みます。

書式 BLOAD "<デバイスディスクリプタ> [<ファイル名>]" [, R] [, <オフセット>]

文例 BLOAD "CAS: マシンゴ", R

(カセットから「マシンゴ」というファイル名の機械語プログラムを読み込み、ただちに実行します。)

解説

ファイルから機械語プログラムをメモリ上に読み込みます。<ファイル名>を省略すると最初に現われたプログラムを読み込みます。

Rをつけると、読み込みが終わった後、ただちに

BSAVEで指定された実行開始番地より実行開始します。<オフセット>を指定するとBSAVEで指定された全てのアドレスが<オフセット>の値だけ移動します。

BSAVE

(ビー セーブ)

機能 メモリ上にある機械語プログラムをファイルに記録します。

書式 BSAVE "<デバイスディスクリプタ> [<ファイル名>]", <先頭番地>, <終了番地> [, <実行開始番地>]

文例 BSAVE "CAS: マシゴ", &HA000,&HAFFF

(&HA000から&HAFFFまでの機械語プログラムに"マシゴ"というファイル名をつけてカセットに記録します。)

解説

メモリ上にある機械語プログラムをファイルに書き込みます。メモリのどこからどこまでをファイルに書き込むかを指定するために<先頭番地>と、<終了番地>を指定します。
<実行開始番地>はBLOAD命令の「R」オプションで

読み込んだ機械語プログラムを実行させるときの、スタート番地を記入します。
また<実行開始番地>を省略すると<先頭番地>が、<実行開始番地>とみなされます。
プログラムは、バイナリ形式で記録されます。

MERGE (マージ)

機能 メモリ上にあるプログラムとカセットテープに記録されているプログラムとを合わせます。

書式 MERGE "<デバイスディスクリプタ> [<ファイル名>]"

文例 MERGE "CAS: SAMPLE"

(カセットから"SAMPLE"というファイル名のプログラムを読み込んで、メモリ上にあるプログラムと合わせます。)

解説

現在メモリ上にあるベーシックプログラムと、カセットテープ上の指定したファイル名のプログラムとを合わせてひとつのプログラムにします。<ファイル名>を省いたときには最初にみつけたものを読み込みます。

プログラム中にMERGE命令があると、実行後、命令待ちに戻ります。

注) MERGEに使うプログラムファイルはSAVEコマンドで記録されたものでなければなりません。メモリ上のプログラムとカセットテープ上のプログラムに同じ行番号があれば、その行番号の内容はテープ上のプログラムの内容に置き換わります。

CLEAR

(クリア)

→ 基礎編 179 ページ

機能 変数値をクリアし、文字領域の大きさとベーシックで使用するメモリの上限を設定します。

書式 CLEAR [〈文字領域の大きさ〉 [, 〈ベーシックで使用するメモリ番地の上限〉]]

文例 CLEAR 300 , &HFFFF

(文字領域の大きさは、300バイト、ベーシックで使用するメモリ番地の上限は、&HFFFFと設定します。)

解説

すべての数値変数を0に、文字変数を「" "」(マルチング)に初期化します。またこのとき開かれていたファイルは全て閉じられます。

パラメータを指定することによって文字領域の大きさと、ベーシックで使用するメモリ番地の上限を設定できます。〈ベーシックで使用するメモリの上限〉を指定する場合、〈文字領域の大きさ〉を省

略することはできません。

電源投入時には〈文字領域の大きさ〉は200バイトに設定されています。

なお、使用できるメモリ番地の上限は&HF380です。

一般^{いっ ばん}ステートメント

LET (レット)

機能 ^{き のう} 変数^{へんすう}に式^{しき}の値^{あたい}を代入^{だいにゅう}します。

書式 ^{しょしき} [LET] ^{へんすうめい} <変数名> ^{しき} = <式>

文例 ^{ぶんれい} LET A=5
(変数^{へんすう} A に 5 を代入^{だいにゅう}します。)

解説 ^{かいせつ}
^{しき} <式> で表される値^{あたい}を変数^{へんすう}や配列^{はいれつ}に代入^{だいにゅう}します。この等号^{とうごう}は代入^{だいにゅう}を表わすものです。「LET」は省略^{しょうりゃく}してもかまいません。
変数^{へんすう}の形^{かたち}と<式>の値^{しき}の形^{あたい} (数値^{すうち}か文字^{もじ}か) が一致^{いっち}していなければなりません。<式>は文字式^{もじしき}の場合^{ばあい}もあります。

サンプルプログラム

```
10 LET A=5
20 LET B$="SAMPLE"
30 PRINT A,B$
40 END
```

(変数^{へんすう} A と B\$ の値^{あたい}を画面^{かめん}表示^{ひょうじ}します。)

REM

(リマーク)

➡ 基礎編 95 ページ

機能 プログラムに注釈を入れます。

書式 REM [<注釈>]

文例 REM これはけいさんプログラムです。

解説

この文から、行の終わりまでは、注釈とみなされ、実行時には無視されます。

REM文の後に「:」(コロン)で区切って実行文を書いても実行されません。

省略形は「,」(アポストロフィ)ですが、省略形で入力した場合には、リストの中でも「,」で表示され、「REM」に変換表示はされません。

FOR～NEXT

(フォー～ネクスト)

➡ 基礎編 103 ページ

機能 FORからNEXTの間のプログラムを指定した回数だけくり返し行ないます。

書式 FOR <変数名>=<初期値> TO <終値> [<STEP><増分>]
:
NEXT [<変数名>] [, <変数名>]…

文例 FOR N=0 TO 10 STEP2
:
NEXT N

(変数Nの値を0から2間隔で10まで変えながら、6回くり返します。)

解説

〈初期値〉、〈終値〉、〈増分〉には数式を用いることもできます。〈増分〉は負の値を用いることもできます。「STEP〈増分〉」を省いた場合は、増分は1となります。

FOR～NEXTは入れ子構造にすることができます。つまり、FOR～NEXTの中にまたFOR～NEXTを置くことができます。このとき1つのFOR～NEXTは完全に他のFOR～NEXTの中になければなりません。FOR～NEXTを重ね、終わる場所が同じであるときは、1つのNEXT文にまとめることができます。その時はNEXTに続けて、内側のFOR～NEXT文の〈変数名〉から順に、「**、**」（カンマ）で区

切って書きます。

```
例  FOR N=1 TO 20
      :
      FOR J=5 TO 10
      :
      NEXT J, N
```

また、NEXT文の〈変数名〉を省略することができます。その場合には、内側のFOR文の変数名とみなされます。

注) FOR文の実行時点で終了条件が成立していても、FOR～NEXTは、必ず1回は実行されます。

GOSUB～RETURN (ゴーサブ～リターン)

➡ 基礎編 163ページ

機能 サブルーチンの呼び出しと復帰を行ないます。

書式 GOSUB 〈行番号〉
RETURN [〈行番号〉]

文例 GOSUB 100

⋮

RETURN 60

(行番号 100 から、始まるサブルーチンに飛び、サブルーチン終了後、行番号60に戻ります。)

解説

GOSUB文により〈行番号〉で示されたサブルーチンに飛び、終了後「RETURN」によりGOSUB文の次の文に戻ります。

RETURN文に〈行番号〉を指定すると、GOSUB文

の次の文へは戻らず、指定した行に戻ります。

サブルーチン中で、CLEAR文が実行されると、RETURN文実行時、「RETURN without GOSUB」のエラーが出ます。

GOTO (ゴートゥー)

➡ 基礎編 96ページ

機能 指定した行番号へ無条件に飛びます。

書式 GOTO 〈行番号〉

GOTO 120

(行番号120へ飛んで命令を実行します。)

かいせつ
解説

指定した〈行番号〉に無条件に実行が移ります。

IF~THEN (ELSE)

イフ～ゼン（エルス）

➡ ^{きそへん}基礎編 122ページ

機能 論理式の値により実行すべき文を選択します。

書式 IF ^{ろんりしき}〈論理式〉 THEN [^{ぶん}〈文〉 [: ^{ぶん}〈文〉…]]
 または ^{ぎょうばんごう}〈行番号〉]
 [ELSE ^{ぶん}〈文〉 [: ^{ぶん}〈文〉…]]
 または ^{ぎょうばんごう}〈行番号〉]

文例 IF A\$="Y" THEN PRINT "YES"
ELSE PRINT "NO"

(文字変数A\$の値がYならYES、それ以外ならNOと画面に表示します。)

かいせつ
解説

^{ろんりしき}論理式^{しせいりつ}が真(成立する)ならば、THEN^{いかぶん}以下の文
 を実行し、ELSE文^{ぶん}があるときはELSE文^{ぶん}の前まで、
 ないときは行の最後までを実行します。もしく論理
 式^{しき}が偽^ぎ(成立しない)ならば、ELSE文^{ぶん}があるとき
 はTHENからELSEまでの文は無視して、ELSE
 以下^{いか}を実行^{じっこう}します。

ない場合には、次の行を実行します。

THENのあとにGOTO文^{ぶん}がくる場合^{ばあい}、THENまたはGOTOのどちらかを省略^{しょうりゃく}することができます。また、ELSE後^ごにGOTO文^{ぶん}がくる場合^{ばあい}には、GOTOを省略^{しょうりゃく}することができます。

ON GOTO/GOSUB

(オン ゴートゥー/ゴーサブ)

➡ 基礎編 174 ページ

機能 式の値により指定した行番号、あるいはサブルーチンへ飛びます。

書式 ① ON <式> GOTO <行番号> [, <行番号>…]
② ON <式> GOSUB <行番号> [, <行番号>…]

文例 ① ON A GOTO 70, 80, 90
(変数Aの値が1のときは行番号70、2のときは80、3のときは90へ飛びます。)
② ON A GOSUB 100, 200
(変数Aの値が1のときは、行番号100のサブルーチンへ、2のときは行番号200のサブルーチンへ飛びます。)

解説

<式>の値はGOTO文またはGOSUB文に続く<行番号>の列の何番目に移るかを示しています。たとえば<式>の値が2であれば、2番目の<行番号>の行に飛びます。式の値が0または指定した<行番号>

<式>の値が255を超える場合は、次の実行可能な文を実行し、式の値が負の数または、255を超える場合はエラーとなります。

STOP

(ストップ)

➡ 基礎編 129 ページ

機能 プログラムの実行を中断します。

書式 STOP

解説

この文が実行されると、プログラムの実行は中断され、コマンド待ちの状態に戻ります。このとき、中断された行番号を、Break in <行番号> と表示します。

このときは、CONT命令により、プログラムの実行を再開することができます。
END文と違って、ファイルは閉じられません。

END

(エンド)

→ 基礎編 83ページ

機能 プログラムの実行を終わさせます。

書式 END

解説

プログラムの実行を終わらせ、すべてのファイルを閉じた後、コマンド待ちに戻ります。END文はプログラム文中にいくつ書いてもかまいません。

プログラムの最後のEND文は省略することができますが、この場合、ファイルは閉じられません。

INPUT

(インプット)

→ 基礎編 112ページ

機能 キーボードから入力します。

書式 INPUT ["〈プロンプト文〉" ;] 〈変数名〉 [, 〈変数名〉…]

文例 INPUT "あなたの なまえは" ; NA\$
(キーボードから文字変数 NA\$ にあなたの名前を入力してください。)

解説

キーボードからの入力を読み、変数に代入します。

INPUT文が実行されると、「？」疑問符を画面に表示して、キーボードからの入力待ちとなります。また〈プロンプト文〉がある場合は、疑問符の前に

その文が表示されます。

データの入力は **RETURN** キーを押すことによって変数に入れます。

LINE INPUT

(ライン インプット)

機能 文字列を1行分、1つの文字変数に入力します。

書式 LINE INPUT [”<プロンプト文>”;] <文字変数名>

文例 LINE INPUT ”なまえ、とし”;N\$
(画面に「なまえ、とし」と表示した後、入力された文字列一行分をN\$に入れます。)

解説

キーボードから入力された、全ての文字データを指定した文字変数に入力します。「一行」は画面上で2～3行にまたがってもかまいません。ただし文字数は255文字以内でなければなりません。入力は[RETURN]キーを押すことによって終了します。<プロンプト文>がある場合、入力を受けつける前に、その文を画面に出力します。INPUT文と違って「?」は画面に出ません。

LINE INPUT 文実行中に [CTRL] + [C] または [CTRL] + [STOP] で実行を中断した後、CONT命令で実行を再開させると、この文の最初より、プログラムを実行します。

サンプルプログラム

```
10 DIM A$(10)
20 FOR N=1 TO 10
30 LINE INPUT A$(N)
40 NEXT N:CLS
50 FOR N=1 TO 10
60 PRINT A$(N)
70 NEXT N
80 END
```

(10個の文字変数に値を入力すると、画面表示されます。)

PRINT

(プリント)

➡ 基礎編 71 ページ

機能 画面に式の値を表示します。

書式 PRINT [<式>[; または ,] <式>…] [; または ,]

文例 PRINT X*256+Y
(X*256+Yの値を画面に出力します。)

解説

文字式や数式の値を画面に表示します。数値を表示する場合、数値の最後にはスペースが1つつけ加えられ、先頭には符号用の1ケタ(正の数の場合スペースとなる)が加えられます。また数値は常に10進数で表示されます。省略形は「?»(疑問符)

です。

PRINT命令の最後に「;」(セミコロン)または「,」(カンマ)をつけた場合、表示後の改行は行なわれません。また、PRINTとだけした場合、1行分のスペースを出力します。

PRINT USING

(プリント ユージング)

機能 フォーマット式に従って文字列や数値を表示する。

書式 PRINT USING "<フォーマット式>" ; <式> [または <式>…]

解説

<フォーマット式>で用いられる書式制御用文字に従って、プリントされる文字や数値の出力書式を決定します。これらの書式制御用文字はそのまま文字として画面に出力されることはありません。

(1) 文字変数や文字列の書式制御

●! (感嘆符)

与えられた文字変数や文字列の最初の文字だけを出力します。

例: PRINT USING " ! is the top. "; "YAMA",
"KAWA", "MORI"

●& (アンド) と _ (スペース)

必ずN個のスペース ($0 \leq N$) が2つの「&」にはさまれた形で用います。このとき、与えられた文字変数や文字列のうち先頭から $(2 + N)$ 個の文字を出力し、あまった文字は無視します。また、指定した長さよりも文字変数や文字列の長さの方が短いときは領域内左づめで出力され、残った部分をスペースで満たします。

例: PRINT USING "&_ _ _ _ _&"; "HITAC
HI", "MB-H1"

●@ (アットマーク)

与えられた文字変数や文字列がそのまま出力されます。

例: PRINT USING "I LIKE @"; "CAT"

(2) 数値表示の書式制御

●# (シャープ) と . (ピリオド)

「#」は数字を表示する桁を、「.」は小数点を表示する位置を示します。したがって出力は固定小数点表示になります。「.」が無い場合は整数

表示です。

指定した桁数よりも数値の桁数の方が短いときは領域内右づめで出力され、上位桁はスペースで満たされます。また小数部分で桁に足りない部分には0が出力されます。

長すぎる場合は、#の桁で4捨5入されます。

符号用として、1つの#が使用されることに注意してください。

例: PRINT USING "###.##_"; 2.1, -2.5,
125.35

●+ と -

数値書式制御文字の並びの前または後ろにつけます。ただし「-」は文字列の後ろにしかつけられません。

「+」をつけたときは数値の前または後ろにその数値の符号が出力されます。「-」をつけたときは負数値の場合のみ負記号を数値の後ろに出力します。

「+」と「-」は、それらを表示するための桁位置を必ず確保することに注意してください。

例: PRINT USING "+###.##.##"; 200,
-25,

●, (カンマ)

小数点位置指定の「.」(ピリオド)の左側に置きます。これを指定した場合は、整数部分を右側から3桁ごとに「,」で区切ります。「.」を表示するたびにそのための桁位置を確保することに注意してください。

例：PRINT USING "#####.,";
3000000

● ** (アスタリスク)

数値書式制御文字の並びの最初に置きます。このとき2桁分の桁位置を確保しますが、数値の上位桁に余白ができると、スペースの代わりにアスタリスク「*」を出力します。

例：PRINT USING "###.##,##.,";
3750.25

● ¥¥ (円記号)

「**」と同様に使いますが、円記号「¥」は数値の直前に1つだけしか出力されません。浮動小数点形式の書式指定に「¥¥」を用いることはできません。負の数の場合、「-」は、¥のすぐ左横にあらわれます。

例：PRINT USING "¥¥#####.,";
10000

● **¥

「**」、「¥¥」と同じように使います。3桁分

の桁位置を余分に確保し、数値の上位に余白領域ができるとその1桁を「¥」で、残りを「*」で満たします。

例：PRINT USING "***###.##";
3.14

● ^^^^ (指数記号)

桁指定の「#」の後ろに置きます。これによって浮動小数点形式の書式が指定され、「#」によって指定するのは仮数部の表示桁となります。

例：PRINT USING "##.####^";
-12.345

注) 表示しようとする数値の桁数が、指定した表示領域を超えてしまった場合は、数値の前に「%」が出力されます。これは四捨五入による丸めが原因となった場合についても同様です。〈フォーマット式〉の中に、上に述べた書式制御用文字以外の文字が現われる場合、それは制御用文字定数とみなしその文字がそのまま出力されます。

LPRINT (エル プリント)

➡ 基礎編 91ページ

機能 プリンタに式の値を出力します。

書式 LPRINT [〈式〉 [または 〈式〉...] [または
また は]]

文例 LPRINT A, B; C\$; D\$

(変数 A、B、C\$、D\$ の値をプリンタに出力します。「,」「;」の違いはPRINT 文と同じ意味です。)

解説

定数、変数や式の値をプリンタに出力します。

〈式〉がない場合は改行のみを行ないます。

PRINT 文と同じように、区切り記号として、「;」、「,」や「」(空白)を使うことができます。

LPRINT 文の最後が「;」で終わっていても、STOP 文や、**CTRL** + **STOP** でプログラムの実行を中断した場合には改行されます。

注) 省略形として、L ? は使えません。

LPRINT USING

(エル プリント ユージング)

機能 ^{き のう} <フォーマット式>^{しき したが}に従って、式^{しき}の値^{あた}をプリンタに出力^{しゅつりょく}します。

書式 ^{しょしき} LPRINT USING " <フォーマット式> "; <式> [; または ; <式> …]

文例 ^{ぶんれい} LPRINT USING "###.##"; 12.5; 1.0E+2; 2.3E-2

解説

プリンタに出力^{しゅつりょく}をするという点^{てん}を除^{のぞ}いては、命令^{めいれい}を参照^{さんしやう}してください。
PRINT USING命令^{めいれい}と同じです。PRINT USING

READ

(リード)

➡ 基礎編 171ページ

機能 ^{き のう} DATA文中^{ぶんちゆう}に書^かかれたデータを読み込み^{よ こ}、変数^{へんすう}に割り当^{わ あ}てます。

書式 ^{しょしき} READ <変数名> [, <変数名> …]

文例 ^{ぶんれい} READ A\$, B\$, C

(DATA文^{ぶん}で定義^{ていぎ}されたA\$, B\$, Cの値^{あたい}を読み込み^{よ こ}それぞれの変数^{へんすう}に代入^{だいりゅう}します。)

解説

DATA文中^{ぶんちゆう}に書^かかれたデータをその順^{じゆん}に、READ文中^{ぶんちゆう}の変数^{へんすう}に割り当^{わ あ}てます。代入^{だいりゅう}すべき変数^{へんすう}が数値変数^{すうちていすう}である場合^{ばあい}、対応^{たいおう}するデータも数値定数^{すうちていすう}でなければなりません。もし、READ文^{ぶん}の変数^{へんすう}の数が、一連^{いちれん}のDATA文^{ぶん}のデータの数^{かず}を超^こえてしまっ

た場合は、Out of DATAエラー^{しやう}が生^はじます。DATA文^{ぶん}のデータが余^{あま}った場合には、残り^{のこ}は無視^{むし}されます。RESTORE文^{ぶん}により、DATA文^{ぶん}のデータを読み直^{よ お}すことができます。

DATA

(データ)

➡ 基礎編 171ページ

機能 READで使用するデータを定義します。

書式 DATA <定数> [, <定数>…]

文例 DATA 1453, 622
(数値1453、622をデータとして定義します。)

解説

DATA文はREAD文で読み込まれる<定数>を用意するための文です。

DATA文は非実行文で、プログラム中に自由に置くことができます。またいくつあってもかまいません。1つのDATA文には、カンマで区切って1

行に入るだけ(255文字以内)の<定数>を書くことができます。READ文は行番号の小さいDATA文から順に<定数>の読み込みを行ないます。

注) DATA文の中の文字列に「,」(カンマ)「.」(ピリオド)や文字列の先頭にスペースを含む場合には、それぞれの文字列全体を「”(ダブルクォート)で囲まなければなりません。

RESTORE

(リストア)

➡ 基礎編 173ページ

機能 READ文で読み込まれるデータの開始行を指定します。

書式 RESTORE [<行番号>]

文例 RESTORE 120
(行番号120のDATA文から、データを読み込みます。)

解説

同じ一群のデータを複数回にわたり読み込む場合などに使用します。RESTORE文が実行されると、次からのREAD文は<行番号>以降にある最初の、

DATA文から読み込みを行ないます。<行番号>を省略した場合、プログラム中最初のDATA文から読み込みを行ないます。

DIM

(ディメンジョン)

→ 基礎編 165ページ

機能 配列の要素の大きさを指定し、メモリ領域を割り合えます。

書式 DIM <変数名> (<添字の最大値> [, <添字の最大値>] ...)

文例 DIM A(30), B\$(5)
(変数Aに31、B\$に6の配列を定義します。)

解説

配列の次元数と添字の最大値を設定します。同時にメモリ上にその配列の領域を割り当てます。配列の次元数はそれぞれの変数の<添字の最大値>の数によって決定されます。添字の最小値は常に0

ですから、1つの配列での要素の数は各次元の、<添字の最大値>に1を加えたものの積となります。DIM文で宣言されない配列変数が使われた場合、添字の最大値は10に設定されます。

ERASE

(イレース)

機能 プログラム中の配列を消去します。

書式 ERASE <配列名> [, <配列名>] ...)

文例 ERASE X, Y, Z
(配列変数X、Y、Zを消去します。)

解説

<配列名>で指定した配列をメモリから消去します。この命令によって、以前配列に使われていた領域を、別の用途に使うことができます。

また、一度ERASEにより消去された配列をDIM文により再び配列宣言することも可能です。

(ERASEされていない配列を2度宣言すると Redimensioned array エラーになります。)

サンプルプログラム

```
10 DIM X(20)
20 FOR N=1 TO 20
30 X(N)=N-2
40 NEXT N
50 ERASE X
60 DIM X(30)
70 FOR N=1 TO 30
80 X(N)=N*3
90 NEXT N
100 END
```

(変数Xの配列を20と宣言していましたが、行番号50で、配列を消去し、行番号60で改めて、配列を30と宣言します。)

DEF FN

(デファイン ファンクション)

→ 基礎編 176 ページ

機能 ユーザ関数の定義をします。

書式 DEF FN <名前> [(<仮引数> [, <仮引数> …])] = <関数の定義式>

文例 DEF FNA (X, Y) = SQR (X * X + Y * Y)
(X * X + Y * Y の平方根を返す FNA (X, Y) という関数を定義します。)

解説

ユーザが定義する式を、FN<名前> という関数として宣言します。その結果他の関数と同様に取り扱うことができます。

仮引数とは、その関数が呼ばれたときに、実際の値と置き換えられる変数です。ここに書かれた変数名と同じ名前の変数名が、他の場所で使われていても、この文は影響を受けません。

<名前> は変数名と同様の条件を満たさなければな

りません。また定義式と名前の型は一致していなければなりません。

DEF FN文は、定義される関数が使用される前に実行されなければなりません。

注) DEF FN文にエラーがあると、エラーメッセージは関数を引用している行を指す場合がありますので注意してください。

DEFINT/SNG/DBL/STR

(デファイン インテジャー/シングル/ダブル/ストリング)

機能 変数名の型宣言を行ないます。

書式 DEF <型> <変数名の頭文字の範囲> [, <変数名の頭文字の範囲> …]

文例 DEFINT I, J
(I, J で始まる変数名が整数型であることを宣言します。)

DEFSNG S, T, V
(変数名がS, T, V で始まる変数が単精度型であることを宣言します。)

DEFDBL D-G
(変数名がDからGまでの頭文字で始まる変数が倍精度型であることを宣言します。)

DEFSTR A, E-F
(変数名がAおよびEからFまでの頭文字で始まる変数が文字型であることを宣言します。)

解説

指定した範囲の頭文字で始まる変数名および配列名の型を宣言します。ただし、変数名に属性文字の「%」整数型、「#」単精度型、「!」倍精度型、「\$」文字型をつけた場合、変数の型は型宣言よりも優先します。

<型>は

INT整数型
SNG単精度型
DBL倍精度型
STR文字型

のいずれかを指定します。

型の宣言を行なわなかった文字で始まり属性文字を持たない変数名の変数はすべて倍精度型とみなされます。

注) DEFと<型>の間は、スペース(空白)をあけると、Syntax errorが出ます。また「-」(マイナス)でつないだ頭文字はアルファベット順でないとSyntax errorが表示されます。

サンプルプログラム

```
10 DEFINT A : DEFSNG B
20 DEFDBL C : DEFSTR D
30 A=6 : B=3.1415927 : C=9
   : D="SAKAI"
40 PRINT A ; B ; C
50 PRINT B/A ; A/C , D
60 END
```

(変数Aを整数型、Bを単精度型、Cを倍精度型、Dを文字型に宣言して、行番号30以下を実行します。)

MID\$ (ミドル ドル)

機能 文字変数の1部を別の文字列で置き換えます。

書式 MID\$ (<文字変数> , <引数1> [, <引数2>]) = <文字式>

文例 MID\$ (A\$, 2, 4) = "WHAT"

(文字変数A\$の左から2文字目からの4文字を "WHAT" に置き換えます。)

解説

<文字変数>の<引数1>番目から<引数2>個の文字列を、<文字式>と交換します。文字は左側から数えることに注意してください。

<引数1>は、<文字変数>の長さ以下の数を指定しなければなりません。<引数2>は0~255の範囲の数が指定できます。

省略すると、<引数1>番目から右を全て与えます。

サンプルプログラム

```
10 A$="I LIKE A DOG"
20 PRINT A$
30 MID$ (A$, 10, 3) = "CAT"
40 PRINT A$
50 END
```

(A\$の文字列の中のDOGをCATに置き換えて表示します。)

SWAP

(スワップ)

機能 ^{き のう} 2つの^{へんすう}変数の^{あた}値を^{こうかん}交換します。

書式 ^{しよしき} SWAP ^{へんすうめい}〈変数名〉, ^{へんすうめい}〈変数名〉

文例 ^{ぶんれい} SWAP A, B
(^{へんすう}変数Aの^{あた}値と、^{へんすう}変数Bの^{あた}値を^{こうかん}交換します。)

解説 ^{かいせつ}
SWAPの^{あと}後に^か書かれた2つの^{へんすう}変数の^{あた}値を^{こうかん}交換します。

^{へんすう}変数の^{かた}型は、^{いっ}一致して^ちいなければなりません。

^{はいれつ}配列^{へんすう}変数も^し使用^{よう}することができます。

サンプルプログラム

```
10 A=10:B=3
20 SWAP A,B
30 PRINT "A=" ;A
40 PRINT "B=" ;B
50 END
```

(^{へんすう}変数Aの^{あた}値と、^{へんすう}変数Bの^{あた}値を^{こうかん}交換して^{がめん}画面に^{りやうじ}表示します。)

にゆうしゆつりよく

入出カステートメント

MAXFILES

(マックス ファイルズ)

機能 同時に開くファイルの最大値を指定します。

書式 MAXFILES = <式>

文例 MAXFILES = 15
(同時に開くファイルの数を15に指定します。)

解説

<式>の値は、0 から15の範囲です。MAXFILES = 0 と指定すると、SAVE,LOADのみが有効となります。

この命令を実行しないときには、開くことのできるファイル数は1です。

OPEN

(オープン)

機能 プログラムで使用できるようにファイルを開きます。

書式 OPEN "<デバイスディスクリプタ> [<ファイル名>]" [FOR<モード>] AS [#]<ファイル番号>

文例 OPEN "CAS: SAMPLE" FOR OUTPUT AS #1
(ファイル番号1のファイルを出力モードで開いて、カセットレコーダのファイル名 "SAMPLE" をプログラムで使用可能にします。)

解説

<デバイスディスクリプタ>で指定したファイルを、指定した<ファイル番号>で開き、ベーシックプログラム中で使える状態にします。

<デバイスディスクリプタ>には以下の4種類が使用できます。「:」も含まれますので注意してください。

CAS: カセットレコーダ

CRT: 画面

GRP: グラフィックスクリーン

LPT: ラインプリンタ

<モード>は以下の3種類があります。

OUTPUT 出力モード

INPUT 入力モード

APPEND 追加モード

<ファイル番号>は1～15の任意の整数を使用できます。ただし、2つ以上を指定するときにはMA XFILES命令で先に宣言しておく必要があります。

CLOSE

(クローズ)

機能 入出力ファイルを閉じます。

書式 CLOSE [[#]<ファイル番号> [, <ファイル番号>...]]

文例 CLOSE #1, 2, 5
(ファイル番号1、2、5のファイルを閉じます。)

解説

指定した番号の入出力ファイルを閉じ、その番号のバッファを他のファイルで使用できるようにしま

す。<ファイル番号>を指定しないでCLOSE文を実行すると、開かれているファイルを全て閉じます。

PRINT

(プリント シャープ)

機能 指定したファイルに式の値を出力します。

書式 PRINT # <ファイル番号>, [<式> [; または <式>...] [; または]]

文例 ① PRINT # 1, "X=" ; X
(番号1のファイルに、"X="の文字列と、変数Xの値を出力します。)

② PRINT # 1, X, Y, Z ;
(番号1のファイルに、変数X、Y、Zの値を出力します。)

解説

<ファイル番号>で指定したファイルに<式>の値を出力します。ファイルはOPEN命令で出力モードに開かれていなければなりません。
ファイルに出力する点を除いてはPRINT命令と同じです。

サンプルプログラム

```
10 OPEN "CAS : SAMPLE" FOR OUTPUT
   AS #1
20 A$="TSUZUKIAOYAMAMAIEKAWA"
30 PRINT #1, A$
40 CLOSE
50 END
60 MAXFILES=2
70 OPEN "CAS : SAMPLE" FOR INPUT
   AS #1
80 B$=INPUT$(7, #1)
90 OPEN "CRT : " FOR OUTPUT AS #2
100 PRINT #2, B$
110 CLOSE
120 END
```

(行番号10～50：カセットレコーダに、行番号20のデータを記録します。行番号60～120：カセットレコーダに記録したデータを読み込み、その内容を画面に表示します。)

PRINT # USING

(プリント シャープ ユージング)

機能 指定したファイルに、フォーマットに従って式の値を出力します。

書式 PRINT # <ファイル番号>, USING "<フォーマット式>" [; <式> [; または <式>...] [; または]]

- 文例** ① `PRINT #1, USING "###.###"; X, Y, Z`
(番号1のファイルに、変数X, Y, Zの値を固定小数点形式で出力します。)
- ② `PRINT #2, USING "###.###^ ^ ^ ^"; 2^20`
(番号2のファイルに、2の20乗の値を浮動小数点形式で出力します。)
- 解説**
 <ファイル番号>で指定したファイルに <フォーマット式> に従って<式>の値を出力します。
- 詳しい説明は、PRINT USING命令、PRINT #命令を参照してください。

INPUT # (インプット シャープ)

- 機能** 入力ファイルからデータを読み込みます。
- 書式** `INPUT # <ファイル番号>, <変数名> [, <変数名>...]`
- 文例** `INPUT #5, A, B`
(ファイル番号5のファイルから、データを読み込み変数A、Bに代入します。)

解説
 指定したファイルからデータを入力します。指定するファイルは、あらかじめ開かれていなければなりません。

サンプルプログラム

```
10 OPEN "CAS : SAMPLE" FOR OUTPUT
   AS #1
20 PRINT #1, 1; 2; 3; 4; 5; 6; 7; 8; 9; 0
30 CLOSE
40 END
50 MAXFILES=2
60 OPEN "CAS : SAMPLE" FOR INPUT
   AS #1
70 OPEN "CRT : " FOR OUTPUT AS #2
80 INPUT #1, A, B, C, D
90 PRINT #2, A; B; C; D
100 CLOSE
110 END
```

(行番号10～40：カセットレコーダに行番号20の数値データを記録します。行番号50～110：記録された数値データを、変数A、B、C、Dに代入し内容を画面に表示します。)

LINE INPUT#

(ライン インプット シャープ)

機能 入力ファイルから1行分のデータを読み込みます。

書式 LINE INPUT# <ファイル番号>, <文字変数名>

文例 LINE INPUT#5, A\$

(ファイル番号5のファイルから、1行分のデータを読み込み、変数A\$に代入します。)

解説

<ファイル番号>で指定したファイルから1行読み込み、変数に代入します。ここでの1行とは改行コードを読み込むまでの入力全てです。ただし、文字変数には255までしか入力できません。

ただし、この場合、前もってCLEAR文で十分、大きく文字領域を確保する必要があります。

改行コードとは、キャラクタコード13(&H0D)の文字、またはキャラクタコードが13(&H0D)の文字と10(&H0A)の文字が連続したものです。

指定したファイルは<ファイル番号>に対して、あらかじめ入力モードで開かれていなければなりません。

サンプルプログラム

```
10 OPEN "CAS : SAMPLE" FOR OUTPUT
   AS #1
20 A$="TSUZUKIAOYAMA"
30 PRINT #1, A$
40 CLOSE
50 END
60 MAXFILES=2
70 OPEN "CAS : SAMPIE" FOR INPUT
   AS #1
80 OPEN "CRT : " FOR OUTPUT AS #2
90 LINE INPUT #1, A$
100 PRINT #2, A$
110 CLOSE
120 END
```

(行番号10～50：カセットレコーダへ文字列データA\$を記録します。行番号60～120：カセットレコーダから、行番号50までで記録したデータを1行分読み込み変数A\$に代入してその内容を画面に表示します。)

INPUT\$

(インプット ドル)

機能 指定した文字数を、指定したファイルから入力します。

書式 INPUT\$ (<文字数>,[#]<ファイル番号>)

文例 INPUT\$ (5, #1)

解説

<文字数>で指定した数の文字を、<ファイル番号>で指定したファイルから読み取ります。指定したファイルは、開かれていなければなりません。

サンプルプログラム

```
10 OPEN "CAS : SAMPLE" FOR OUTPUT
   AS #1
20 A$="6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
   3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 0
   0 0 "
30 PRINT #1, A$
40 CLOSE
50 END
60 MAXFILES=2
70 OPEN "CAS : SAMPLE" FOR INPUT
   AS #1
80 B$=INPUT$(15, #1)
90 OPEN "CRT : " FOR OUTPUT AS #2
100 PRINT #2, B$
110 CLOSE
120 END
```

(行番号10～50:カセットレコーダに、行番号20のデータを記録します。行番号60～120:カセットレコーダから、行番号50までのプログラムで記録したデータのうち15文字分を入力し、画面に表示します。)

画面・グラフィックステートメント

PSET 命令、PRESET 命令、LINE 命令、CIRCLE 命令、PAINT 命令の座標（画面位置）には、相対的位置を表わすSTEPも使えます。226ページ、149ページのPUT SPRITE 命令をご覧ください。

CLS (クリア スクリーン)

➡ 基礎編 78 ページ

機能 画面をクリアし、カーソルをホーム位置へ移動させます。

書式 CLS

解説

全ての画面モードでその表示内容を消去し、カーソルをホーム位置（画面左上隅）に置きます。

LOCATE (ロケイト)

➡ 基礎編 101 ページ

機能 カーソルの位置および働きを指定します。

書式 LOCATE <水平位置>, <垂直位置> [, <カーソルモード>]

文例 LOCATE 15, 12

（カーソルを、水平位置15、垂直位置12の場所に移動します。）

解説

カーソルを指定した位置へ移動します。位置はテキストモードのときの画面の位置です。39×24テキストモードの場合、〈水平位置〉の範囲は0～38、〈垂直位置〉の範囲は0～22(KEY OFFで23)です。29×24テキストモードの場合、〈水平位置〉の範囲は0～28、〈垂直位置〉の範囲は0～22(KEY OFFで

23)です。なお、原点(0, 0)は左上隅です。〈カーソルモード〉で0を指定するとカーソルを表示しません。ただしプログラム実行中でしか使用できません。1を指定すると、カーソルを表示します。省略すると、1が設定されます。

WIDTH (ウィドス)

機能 1行に表示する文字を設定します。

書式 WIDTH 〈表示文字数〉

文例 WIDTH 25
(画面表示文字数を1行25文字にします。)

解説

画面1行に表示する文字数を〈表示文字数〉に設定します。39×24テキストモード(SCREEN 0)では1～40、29×24テキストモード(SCREEN 1)では1～32の数字が指定できます。

注 標準モード(SCREEN 0で39文字、SCREEN1で29文字)より多い表示文字数を設定すると、文字が画面からはみ出ることがあります。

COLOR (カラー)

➡ 基礎編 131 ページ

機能 文字色と背景色と境界色を指定します。

書式 COLOR [〈文字色コード〉] [, 〈背景色コード〉] [, 〈境界色コード〉]

文例 COLOR 11, 8, 15
(文字色を明るい黄色に、背景色を赤に、境界色を白に設定します。)

かいせつ 解説

がめんぜんたいのふじいろ はいけいぜんたいのいろ してい
画面全体の文字の色と背景全体の色を指定します。

くふじいろコード くはいけいいろコード くきょうがいいろコード

とも 0～15で指定します。カラーコードの対応は
つぎ
次のとおりです。

- | | |
|---|------------|
| 0 | とうめい
透明 |
| 1 | くろ
黒 |
| 2 | ろく
緑 |
| 3 | あか
明るい緑 |
| 4 | くら
暗い青 |
| 5 | あか
明るい青 |
| 6 | くら
暗い赤 |

- | | |
|----|------------|
| 7 | みずいろ
水色 |
| 8 | あか
赤 |
| 9 | あか
明るい赤 |
| 10 | くら
暗い黄 |
| 11 | あか
明るい黄 |
| 12 | くら
暗い緑 |
| 13 | むらさき
紫 |
| 14 | はい
灰 |
| 15 | しろ
白 |

でんげんとうにゅうじ
電源投入時には、15、4、7がそれぞれ設定されています。
しょうりやく
省略した場合、以前に指定されたカラーコードと同じとみなします。

SCREEN (スクリーン)

➡ 基礎編 133 ページ

きのう
機能 画面モード、スプライトの大きさ、キーのクリック音の有無、カセットレコーダへの
ボーレイト、せんようプリンタの有無を指定します。

しょしき
書式 SCREEN [くがめんモード] [, <スプライトサイズ>] [, <キー クリック>] [, <ボ
ーレイト>] [, <プリンタ オプション>]

ぶんれい
文例 SCREEN 2 , 3 , 1 , 1 , 1

(画面モードをハイリゾリューションモードに設定し、スプライトサイズ16ドット×16ドットの拡大
表示、キーを押したときにクリック音を発生させます。ボーレイトは1200ボーモード、使用プリン
タがMSX専用以外のものを指定します。)

解説

①<画面モード>は0～3の数字が指定できます。

- 0 : 39文字×24行 テキストモード
- 1 : 29文字×24行 テキストモード
- 2 : 256ドット×192ドット ハイリゾリューションモード
- 3 : 256ドット×192ドット マルチカラーモード

●テキストモード(画面モード0、1)ではグラフィック関係の命令を実行するとエラーになります。必ずSCREEN命令先に実行してください。

●また、INPUT文を実行したとき、および、コマンド待ちの状態になったときにはテキストモードに変更されます。

省略すると、1とみなします。

②<スプライトサイズ>は0～3の数字が指定できます。

- 0 : 8ドット×8ドット標準
- 1 : 8ドット×8ドット拡大
- 2 : 16ドット×16ドット標準
- 3 : 16ドット×16ドット拡大

●この命令を実行するとSPRITE\$で設定したスプライトパターンは消えてしまいます。

省略すると、電源投入時では0を、または、以前に指定したサイズとみなします。

③<キー クリック>はキーを押したときのポツ、

ポツというクリック音発生の有無を指定します。

0、1が指定できます。

0 : クリック音を発生しない

1 : クリック音を発生させる

省略すると、電源投入時では1を、または、以前に指定した値とみなします。

④<ボーレート>はカセットレコーダへの書き込み速度を指定します。1、2の2種類あります。

1 : 1200ボー

2 : 2400ボー

省略すると、電源投入時では1を、または、以前に指定したボーレートとみなします。

●カセットレコーダから読み込む場合、ボーレートは自動的に決定されますので、書き込んだ時のボーレートの値を記録しておく必要はありません。

⑤<プリンタ オプション>は使用するプリンタがH1専用のものか、そうでないかを指定します。

0以外の数字を指定すると、プリンタはMSX専用ではないとみなされ、グラフィックキャラクターはスペースに、ひらがなはカタカナに換えて、プリンタに出力されます。

省略すると、電源投入時では0を、または、以前に指定した値とみなします。

注) SCREEN 0の場合、ひらがなが少し欠ける場合があります。

PSET

(ピー セット)

➡ 基礎編 135 ページ

機能 指定した位置に点を打ちます。

書式 PSET (<水平位置, 垂直位置>) [, カラーコード]

文例 PSET (50, 50), 6

(グラフィック画面上50, 50の位置に暗い赤で点を打ちます。)

解説

画面上に、〈カラーコード〉で指定した色で点を打ちます。

常にグラフィック画面で使用するために、〈水平位置〉は、0～255まで、〈垂直位置〉は、0～191までの範囲となります。〈カラーコード〉は、COLOR 命

令で説明した0～15の数字が使えます。省略した場合、電源投入時に決められている文字色、またはその後COLOR 命令で指定された文字色が選択されます。

注) テレビ信号の特性により、指定した色と異なることがあります。

PRESET (ピー リセット)

➡ 基礎編 137 ページ

機能 指定した位置の点を消します。

書式 PRESET (〈水平位置〉, 〈垂直位置〉) [, 〈カラーコード〉]

文例 PRESET (100, 20)

(グラフィック画面上、100, 20の位置に打たれている点を消します。)

解説

〈カラーコード〉を省略した場合、指定した画面上の位置にある点を消します。

〈カラーコード〉をつけたときは、PSET 命令と同じ機能を持つことになります。

LINE (ライン)

➡ 基礎編 138 ページ

機能 指定した2点間に直線を引きます。また、その2点を対角とする長方形を描いたり、色で塗りつぶしたりします。

書式 LINE [(〈始点位置〉) - (〈終点位置〉) [, 〈カラーコード〉] [, 〈オプション〉]

文例 ① LINE (20, 30) - (100, 100)

(座標 20, 30 から 100, 100 まで文字色で線を引きます。)

② LINE (10, 10) - (100, 100), 8, B

(座標 10, 10 と 100, 100 を対角とする長方形を、赤で描きます。)

③ LINE (0, 0) - (255, 191), 1, BF

(座標 0, 0 と 255, 191 を対角とする長方形を黒で描き、中を同色で塗りつぶします。)

解説

〈始点位置〉から〈終点位置〉まで〈カラーコード〉で指定した色で直線を引きます。〈オプション〉にBを指定すると〈始点位置〉と〈終点位置〉を対角とする長方形を描き、BFを指定すると長方形の中を指定したカラーコードの色で塗りつぶします。〈カラーコード〉が省略された場合は、COLOR

命令で指定された文字色となります。

〈始点位置〉が省略された場合は、以前に指定した〈終点位置〉とみなします。電源投入時〈始点位置〉は、(0, 0)に設定されています。

注) たて線の場合、テレビ信号の特性により、指定した色と異なる場合があります。

CIRCLE

(サークル)

➡ 基礎編 142 ページ

機能 円、楕円、弧を描きます。

書式 CIRCLE (〈中心位置〉), 〈半径〉[, 〈カラーコード〉][, 〈開始角度〉][, 〈終了角度〉][, 〈扁平率〉]

文例 CIRCLE (120, 100), 30, 1, 0, 6.28, 0.5
(座標120, 100を中心に、横方向の半径30で、横長のだ円を黒で描きます。)

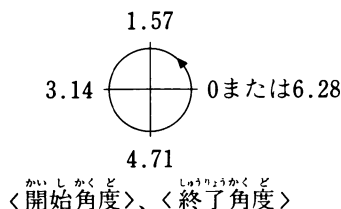
解説

〈中心位置〉を中心に、〈半径〉で指定される大きさの円を描きます。〈カラーコード〉が、省略された場合には、COLOR 命令で指定された文字色となります。〈開始角度〉、〈終了角度〉はラジアンで指定します。また〈開始角度〉、〈終了角度〉が負の数値で指定されると〈開始角度〉と中心と〈終了角度〉を結ぶ扇形を描きます。ただし、〈開始角度〉、〈終了角度〉を-0と指定しても、中心とは結ばれません。

だ円を書く場合の〈扁平率〉は、1より小さい場合横長のだ円、1より大きい場合、たて長のだ円となり、〈半径〉は、〈扁平率〉が1より小さい場合は横方向の長さ、1より大きい場合はたて方向の長さとなり、省略すると1とみなします。

〈開始角度〉は省略すると0とみなします。

〈終了角度〉は省略すると6.28とみなします。



PAINT

(ペイント)

➡ 基礎編 140 ページ

機能 指定された境界の中を塗りつぶします。

書式 PAINT (〈開始位置〉) [, 〈カラーコード1〉] [, 〈カラーコード2〉]

文例 ① PAINT (100, 51), 8, 1

(黒で囲まれた図形の中を赤で塗りつぶします。座標100, 51が図形の外にある場合は図形の外側を赤で塗りつぶします。(マルチカラーモードの場合のみ))

② PAINT (50, 80), 3

(明るい緑の図形を同じ色で塗りつぶします。この場合、<カラーコード2>は省略できます。(ハイリゾリューションモード))

解説

<カラーコード2>の色で囲まれた図形の中または外を<カラーコード1>の色で塗りつぶします。開始位置が図形の内側にある場合には図形の中を、図形の外側にある場合には、図形の外側を塗りつぶします。

イ) ハイリゾリューションモード (SCREEN 2) では<カラーコード1>と<カラーコード2>は同じでなければなりません。

この場合、<カラーコード2>は省略できます。<カラーコード1>は省略すると、文字色と同じとみなします。

ロ) マルチカラーモード (SCREEN 3) では、<カラーコード1>と<カラーコード2>は別々に指定できます。<カラーコード1>を省略すると文字色と同じとみなします。<カラーコード2>を省略すると<カラーコード1>と同じとみなします。

PUT SPRITE (プット スプライト)

➡ 基礎編 148 ページ

機能 指定したスプライトを画面に表示します。

書式 PUT SPRITE <スプライト面番号> [, <表示位置>] [, <カラーコード>] [, <パターン番号>]

文例 PUT SPRITE 0, (100, 100), 8, 1
(パターン番号1をスプライト面0の(100, 100)の画面位置に赤で表示します。)

解説

<パターン番号>で指定したスプライトを<スプライト面番号>で指定したスプライト面の<表示位置>に<カラーコード>の色で表示します。

<スプライト面番号>は0～31の値とれます。

<表示位置>には次の2通りの書き方があります。

i) STEP (<x方向の増分>, <y方向の増分>)

現在の表示位置から相対的にスプライトを移動させたい場合に用います。

ii) (<x位置>, <y位置>)

スプライトを画面の絶対位置に表示させる場合に用います。

<x位置>は-32～255、<y位置>は-32～191の範囲で使用します。<y位置>に208が指定された場合、指定されたスプライト面の背後にあるすべてのスプライトは画面に表示されなくなります。これは、208以外の値が、そのスプライトに設定されるま

で続きます。

また、<y位置>に209を指定した場合、指定されたスプライトが画面に表示されなくなります。

<表示位置>を省略した場合、以前指定した同じスプライト面の位置とみなします。指定していなければ画面には表示されなくなります。

<カラーコード>は COLOR 命令で使われるものと同じです。省略すると文字色と同じになります。

<パターン番号>は SPRITE\$ 命令で作成したスプライトの番号を使用します。スプライトサイズ (SCREEN 命令で設定します) が0か1の場合は0～255まで、2、3の場合は0～63までが使用できます。また、<パターン番号>を省略した場合は、<スプライト面番号>と同じとみなされます。

DRAW

(ドロー)

機能 画面上に図形を描きます。

書式 DRAW "<文字列>"

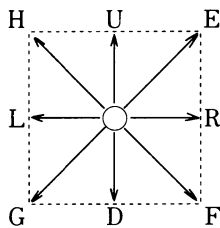
解説

作図命令を使って、画面上に図形を描きます。作図命令は15種類あり、文字で表わされます。それらを組み合わせて図形を描きます。

ここで使われる作図命令には次のようなものがあります。

●移動命令

U <距離> 上へ移動
D <距離> 下へ移動
L <距離> 左へ移動
R <距離> 右へ移動
E <距離> 右斜め上へ移動
F <距離> 右斜め下へ移動
G <距離> 左斜め下へ移動
H <距離> 左斜め上へ移動



Mx, y 移動位置をx方向、y方向各々についての絶対座標あるいは相対座標で指定します。ここで、xの前にプラス(+)あるいはマイナス(-)の符号が書かれていれば、相対座標指定と解釈されます。

移動命令は、基準点からそれぞれの方向に指定された距離だけ作図します。各命令の実行後の基準点は描き終わった最後の点となります。各方向への移動距離は、上の各命令中の<距離>にS命令で決められる倍率をかけたものになります。

なお、各命令に同じ<距離>を指定しても、画面上の実際の距離は、上図のように作図方向により異なります。距離の単位は、ドットです。

●その他の命令

B 各々の方向へ移動しますが、作図は行ないません。

N 各々の方向へ作図しながら移動しますが、作図後の基準点は始点となります。

A<角度> U、D、L、R、E、F、G、Hおよび相対座標指定のときのM命令で作図した図形を90°単位で反時計方向に回転させて表示します。一度指定すると次のA命令まで有効となります。ここで<角度>は0～3の数字で表わされ、0は0度、1は90度、2は180度、3は270度を表わします。省略すると0とみなします。

C<カラーコード>

図形に色をつけます。<カラーコード>は0～15の値を指定します。

省略すると、文字色と同じとみなします。

S<倍率係数> 倍率を決めます。

<倍率係数>の範囲は0～255の整数です。この値の4分の1が倍率となります。たとえば数値に1を指定した場合、倍率は4分の1となります。ただし0は4と同じで、倍率1となります。U、D、L、R、E、F、G、H命令およびM命令の相対指定で指定された距離に倍率がかけ合わされて、移動・作図時の距離が決められます。<倍率係数>の指定を省いた場合は0が設定されます。S命令は指定すると、次のS命令まで有効となります。

X<文字変数>; <文字変数>で示される作図命令を実行します。X命令を使用することにより、部分的な図形を定義することができます。

以上の命令中、<距離>や<カラーコード>などには、定数の他に「=<変数名>;」の形式で数値変数を指定することができます。

サンプルプログラム

```
10 SCREEN 2
20 A$ = "U80R80D80L80"
30 DRAW "BM80, 150XA$;"
40 GOTO 40
50 END
```

(80, 150) を1つの頂点とする1辺の長さ80の長方形を描きます。

おん がく えん そう

音楽演奏ステートメント

BEEP (ビーブ)

→ 基礎編 159 ページ

機能 き のう スピーカーから音を出します。おと だ

書式 しょしき BEEP

解説 かいせつ

PRINT CHR\$(7)は同じ機能です。おな き のう

PLAY (プレイ)

→ 基礎編 159 ページ

機能 き のう 音楽を演奏します。おん がく えん そう

書式 しょしき PLAY <文字式 1> [, <文字式 2> [, <文字式 3>]]
も じ しき も じ しき も じ しき

文例 ぶんれい PLAY "CDEFGAB", "DEFGABC", "EFGABCD"

(「ドレミファソラシ」と「レミファソラシド」と「ミファソラシドレ」を同時に演奏します。)
どう じ えん そう

解説

〈文字式1〉、〈文字式2〉、〈文字式3〉で指定された音楽を演奏します。3つの文字式で、3重の和音を出力することができます。

文字式に使われる記号の意味は次のとおりです。

- i) A、B、C、D、E、F、G、#、+、-
現在指定されているオクターブで、音を出します。A～Gはそれぞれ

C D E F G A B

ド レ ミ ファ ソ ラ シ

を表わします。これらの文字の後に書かれた

「#」、「+」、「-」はそれぞれ

#、+…シャープ(半音上げる)

- …フラット(半音下げる)

を意味し、ピアノの黒い鍵盤に相当します。

ただし、黒鍵に相当しないC-はB、E+はF、

F-はE、B+はCと同オクターブの別のキー

に対応します。

- ii) O〈数字〉

Oの後に書いた〈数字〉でオクターブを指定します。1～8の数字で8オクターブ指定できます。Oが指定されないときは、以前のオクターブ値を保持します。電源を入れたときにはO4に設定されています。

- iii) N〈数字〉

Nの後に書かれた〈数字〉でA～G、#、+、-の代わりに音階を指定します。0～96の数字が使えますが、0は休符を指定したことになります。

O 1

C	C+	D	D+	E	F	F+	G	G+	A	A+	B
D-	D-	E-	F-	G-	A-	B-					
1	2	3	4	5	6	7	8	9	10	11	12

O 4

C	C+	D	D+	E	F	F+	G	G+	A	A+	B
D-	D-	E-	F-	G-	A-	B-					
36	37	38	39	40	41	42	43	44	45	46	47

- iv) L〈数字〉

Lの後に書かれた〈数字〉で、次から演奏する音の長さを指定します。1～64の数字を使うことができます。電源投入時の値は4です。

L 1 : 全音符

L 2 : 2分音符

⋮

L 4 : 4分音符

⋮

L 8 : 8分音符

⋮

L 64 : 64分音符

- v) R〈数字〉

休符です。〈数字〉の数だけ演奏を休みます。

〈数字〉の意味は「L」と同じです。

- vi) ・(ピリオド)

音の長さを表わす数字の後に付けると、符点音符を意味するようになります。R(休符)の後に付けることもできます。

- vii) T〈数字〉

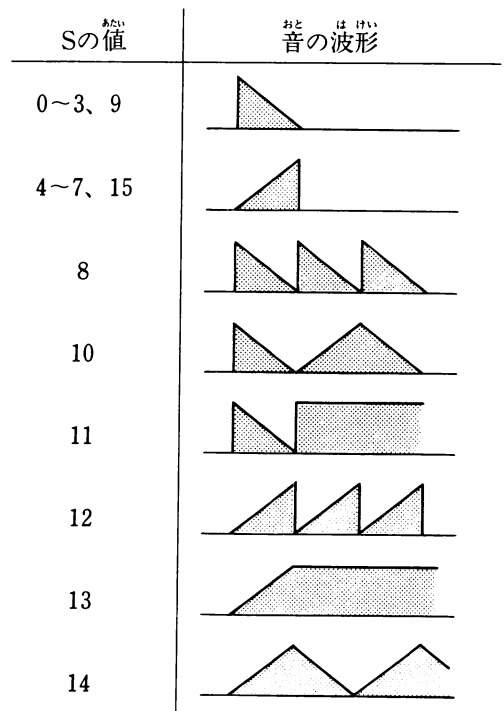
〈数字〉で曲全体のテンポを指定します。〈数字〉は1分間に演奏される4分音符の数です。32～255までが指定できます。電源投入時の値は120です。

- viii) V〈数字〉

音の大きさを指定します。〈数字〉は0～15の範囲で、0は音を出しません。電源投入時の値は8です。〈数字〉は13くらいが適当です。

- ix) S〈数字〉

出力波形の形状を指定します。〈数字〉は1～15が指定でき、次の図に示すような波形を出力します。電源投入時の値は1です。



指定を解除するときにはV<数字>を使ってください。もとの波形に戻ります。したがってVとSは同時に指定できません。M<数字>といっしょに用いて、音色を変化させるのに使います。

x) M<数字>
<数字>でエンベロープの周期(次ページ参照)を指定します。1～65535までの数字が設定で

きます。電源投入時の値は255です。

X<変数>;
文字変数をPLAY文中で使う場合、X<変数>;の形で入れることができます。

以上の命令中 <数字> のところは定数の他に、
「=<変数名>;」の形式で数値変数を指定することができます。

SOUND

(サウンド)

機能 指定したPSG(サウンドIC)のレジスタに直接データを書き込み、音を発生させます。

書式 SOUND <レジスタ番号>, <データ>

解説

PSGのR0～R13まで14個のレジスタに直接データを書き込むことによって、PLAY命令では出すことのできない複雑な音を発生させることができます。

レジスタ		ビット							
		B7	B6	B5	B4	B3	B2	B1	B0
R 0	チャンネルA周波数	8BIT				FT A			
R 1	チャンネルA周波数					4BIT CT A			
R 2	チャンネルB周波数	8BIT				FT B			
R 3						4BIT CT B			
R 4	チャンネルC周波数	8BIT				FT C			
R 5						4BIT CT C			
R 6	ノイズ周波数					5BIT NP			
R 7	チャンネル設定	IN/OUT		NOISE			TONE		
		1	0	C	B	A	C	B	A
R 8	チャンネルA音量					M	L3	L2	L1
R 9	チャンネルB音量					M	L3	L2	L1
R10	チャンネルC音量					M	L3	L2	L1
R11	エンベロープ周期	8BIT FT							
R12		8BIT CT							
R13	エンベロープ形状					E3	E2	E1	E0

i) R0～R5
発生させたい音の周波数を設定します。PSGには3チャンネルのトーンジェネレータがあるので、R0、R1でチャンネルA、R2、R3でチャンネルB、R4、R5でチャンネルCを指定します。

ここではチャンネルAのR0、R1について説明します。出力したい周波数を f_T とした場合

$$\frac{1.7898 \times 10^6}{16 \times f_T}$$

の上位1バイトがR1に、下位1バイトがR0に入ります。

例) $f_T = 200\text{Hz}$ の場合

$$\frac{1.7898 \times 10^6}{16 \times 200} \div 559$$

559は&H22Fですから、R1には&H2、R0には&H2Fを書き込めばよいことになります。プログラムでは、

SOUND 1, &H2

SOUND 0, &H2F

を実行すればよいわけです。

R2～R5についても同様です。

ii) R6

ノイズ周波数を設定します。出力したいノイズ周波数を f_N とすると、

$$\frac{1.7898 \times 10^6}{16 \times f_N}$$

をR6に入れます。

例) $f_N = 10\text{kHz}$ の場合

$$\frac{1.7898 \times 10^6}{16 \times 10 \times 10^3} \div 11$$

従^{したが}ってR6には11を書^かき込^こめばよいわけです。

プログラムでは

SOUND 6, 11

を実行^{じっこう}します。

iii) R7

使用^{しよう}したいチャンネル(A、B、C)およびノイズ発生^{はっせい}の有無^{うむ}を指定^{ししてい}します。使用^{しよう}したいチャンネルのビットを0にしたデータを書^かき込^こむことで指定^{ししてい}できます。

例^{れい}) チャンネルAのトーンとA、Bのノイズを使う^{つか}場合^{ばあい}、

B7	B6	B5	B4	B3	B2	B1	B0
1	0	1	0	0	1	1	0

となるので&B10100110(&HA6)をR7に書^かき込^こめばよいわけです。プログラムでは

SOUND 7, &B10100110

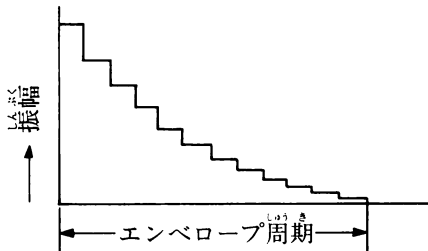
を実行^{じっこう}してください。

注) B7=1、B6=0 と指定^{ししてい}してください。

iv) R8~R10

A、B、C各チャンネルの音量^{おんりょう}を設定^{せってい}します。下位4ビットに0~15の数値を書^かき込^こむことによって音量^{おんりょう}を変^かえることができます。0のときが無音^{むおん}、15が最大^{さいだい}です。

また、これらのレジスタのB4を1に設定^{せってい}すると音量^{おんりょう}が時間^{じかん}的に変^{へん}化するモード(エンベロープ・モード)にすることができます。この場合^{ばあい}、どのような形状^{けいじょう}でエンベロープを出力^{しゅつりょく}するのか、またその周期^{しゅうき}はどのくらいになるのかをR11~R13で指定^{ししてい}しなければなりません。



例^{れい}) チャンネルAの音量^{おんりょう}を15に、Bの音量^{おんりょう}を7にし、チャンネルCをエンベロープ・モードで使^{つか}いたい。

SOUND 8, 15

SOUND 9, 7

SOUND 10, 16

v) R11~R12

エンベロープ周期^{しゅうき}を設定^{せってい}します。エンベロープ周期^{しゅうき}を f_E とするとき、

$$\frac{1.7898 \times 10^6}{256 \times f_E}$$

の上位1バイトをR12に下位1バイトをR11に書^かき込^こみます。

例^{れい}) $f_E = 0.2\text{Hz}$ の場合^{ばあい}

$$\frac{1.7898 \times 10^6}{256 \times 0.2} \div 35000$$

35000は&H88B8ですから、R12には&H88、R11には&HB8を書^かき込^こめばよいことになります。プログラムでは、

SOUND 12, &H88

SOUND 11, &HB8

を実行^{じっこう}すればよいのです。

vi) R13

下位4ビットでエンベロープの形状^{けいじょう}を設定^{せってい}します。E0からE3の値がどのような波形^{はけい}に対応^{たいおう}しているのかを下図^{かづ}に示^{しめ}します。なお、図中の×は0でも1でもかまいません。

E3	E2	E1	E0	対応 ^{たいおう} する10進数 ^{しんすう}
0	0	×	×	0~3
0	1	×	×	4~7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

例^{れい}) 上図^{じやうず}の上から3番目^{うへ}の波形^{はけい}でエンベロープを出力^{しゅつりょく}したい。

SOUND 13, &B1000

サンプルプログラム

```
10  FORN=0 TO 15
20  SOUND 0, &H18
30  SOUND 1, &H1
40  SOUND 7, &HFE
50  SOUND 8, N
60  SOUND 11, &H46
70  SOUND 12, &H0
80  SOUND 13, &H8
90  FOR J=1 TO 100
100 NEXT J, N
110 END
```


(PSGのチャンネルAを^{つか}って^{おと}音を^{はっせい}発生させます。)

ファンクションキー^{かん けい}関係ステートメント

KEY (キー)

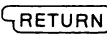
機能 ファンクションキーの内容^{ないよう}を決^きめます。

書式 KEY <ファンクションキー^{ばんごう}番号>, <文字^{も じ}式^{しき}>

文例 KEY 6, "CLS"+CHR\$ (13)
(ファンクションキー F6 にCLS  を定^{てい}義^ぎします。)

解説

ファンクションキーに文字列^{も じ れつ}を定^{てい}義^ぎします。<ファンクションキー^{ばんごう}番号>は1から10までです。1つのキーには最大15文字の文字列^{も じ れつ}が定^{てい}義^ぎできます。16文字以上定^{てい}義^ぎしようとしても、残^{のこ}りは無^む視^しされます。ただし画面表示^{が け ん ひょう し}は、5文字^{も じ}までです。

 (キャラクターコード13)や、「[”]」(キャラクターコード34)はCHR\$関^{かん}数^{すう}を使^{つか}って定^{てい}義^ぎします。(付録のコントロールコード一覧表^{いちらんひょう}を参^{さん}照^{しょう}してください。)電^{でん}源^{げん}を切^きると設^{せつ}定^{てい}は初^{しよ}期^き状^{じょう}態^{たい}に戻^{もど}ります。

KEY LIST

(キー リスト)

機能 ファンクションキーの定義内容を画面に表示します。

書式 KEY LIST

解説

ファンクションキーに定義されている文字列を、
F1からF10まで順に画面に表示します。

注) F10のRUNは、F5のRUNと違い、CLS命令
のコントロールコードが入っているため、先
頭にスペースが表示されます。

KEY ON/OFF

(キー オン/オフ)

機能 ファンクションキーに定義される文字列を表示します。

書式 ①KEY ON
②KEY OFF

解説

①KEY ON

テキスト画面にファンクションキーの表示を行な
っているため、この状態では最下行は使用できま
せん。
電源投入時はこの状態です。

②KEY OFF

この命令を実行させるとファンクションキーの表
示が消え、最下行も使用可能となります。
ファンクションキーからの入力はできます。

エラー^{しよ り}処理ステートメント

ERROR (エラー)

機能 エラーを^{も ぎ て き}模擬的に^{はっ せい}発生させます。

書式 ERROR <エラーコード>

文例 ERROR 1 (エラーコード1に^{たいおう}対応するエラーを^{も ぎ て き}模擬的に^{はっ せい}発生させます。)

解説

<エラーコード>で^{してい}指定したエラーが^{はっ せい}発生した状態を^{も ぎ て き}模擬的に^{つく}作りだします。<エラーコード>は1～255の^{せいすう}整数でなければなりません。

エラーコードのうち、26～49および60～255は、^み未^{てい ぎ}定義のエラーコードでこれらの^か値を^し指定すると、「Unprintable error」を^{しゅつりょく}出力します。

エラーコードについては、^{ふろく}付録のエラーメッセージ^{いち らん ぶ じょう}一覧表を^{さんしやう}参照してください。

ON ERROR GOTO

(オン エラー ゴートゥー)

機能 エラーが発生したとき、指定した行番号に実行を移します。

書式 ON ERROR GOTO <行番号>

文例 ON ERROR GOTO 100

(プログラムの実行中にエラーが発生した場合は、行番号100へ飛びなさい。)

解説

ON ERROR GOTO文を前もって実行しておくと、プログラム中で何かのエラーが生じたときに、指定した<行番号>に実行を移します。これにより、エラーの発生によるプログラムの実行の中断をふせぐことができます。

<行番号>に0を指定してメインプログラムで実行するか、またはCLEAR命令を実行すると以上の働きをなくします。エラー回復処理ルーチン中で実行すると、エラー回復処理を中止し、エラーメッセージを画面に表示してプログラムの実行を中断します。通常は、RESUMEで戻ります。

注) 命令待ちの状態が発生したエラー、およびエラー回復処理ルーチンを実行中に発生したエラーに対しては効力がありません。

また、INPUT文実行中にキー入力された内容によって発生するエラーは、再び入力待ちの状態となるため、エラー処理は行ないません。<行番号>に0を指定しても、ON ERROR GOTO 0として動作するため、行番号0をエラー処理ルーチンの開始行に指定することはできません。

RESUME

(リジューム)

機能 エラー処理終了後、指定した場所から実行を再開します。

書式 ① RESUME [<行番号>]

② RESUME NEXT

解説

エラー処理プログラムの実行後、メインプログラムの実行を再開します。実行を再開する場所に依りて3つの書式を選ぶことができます。

① RESUME <行番号>

<行番号>の行から実行が再開されます。

ただし、<行番号>に0を指定した場合は、エラーの生じた文から実行が再開されます。

② RESUME NEXT

エラーの生じた文の次の文から実行が再開されます。

③ RESUME

エラーの生じた文から実行が再開されます。

RESUME文はエラー処理プログラムの終わりを示すもので、メインプログラムへ復帰しない場合を除いて必ず置く必要があります。

サンプルプログラム

```
10 ON ERROR GOTO 100
20 A=6
30 FOR B=3 TO -3 STEP-1
40 C=A/B
50 PRINT A; "/ "; B; " = "; C
60 NEXT B
70 END
100 PRINT A; "/ "; B; " = とけません"
110 RESUME 60
```

(エラー(0で割る)が発生した場合、100行目に飛び、「= とけません」と表示して、60行目に戻ります。)

わりこしより 割り込み処理ステートメント

ON INTERVAL GOSUB (オン インターバル ゴーサブ)

機能 行番号で指定したインターバルタイマ割り込みルーチンへ飛びます。

書式 ON INTERVAL=じかん<時間> GOSUB ぎょうばんごう<行番号>

文例 ON INTERVAL=60 GOSUB 100
(プログラム実行中、1秒ごとにぎょうばんごう行番号100のサブルーチンを実行します。)

解説

じかん<時間>に書かれた数値の1/60秒ごとに、ぎょうばんごう<行番号>で始まる割り込みルーチンを実行します。

この文を有効とするためには、以前にINTERVAL ON 命令を実行しておく必要があります。

また割り込みルーチンでは、タイマ割り込みは受け付けなくなります。

注) ベーシック・プログラムの実行中でないと割り込みは発生しません。

また、ON ERROR文と同時に使用することはできません。

INTERVAL ON/OFF/STOP

(インターバル オン/オフ/ストップ)

機能 インターバルタイマによる割り込みの発生を許可、禁止、または停止します。

書式

- ①INTERVAL ON
- ②INTERVAL OFF
- ③INTERVAL STOP

解説

①INTERVAL ON

新しい命令文を実行するたびに、インターバルタイマをチェックし、指定された時間になると、ON INTERVAL GOSUB<行番号>で指定した割り込みを発生させます。

②INTERVAL OFF

割り込みの発生は禁止され、インターバルタイマのチェックも行ないません。

③INTERVAL STOP

割り込みの発生は禁止されますが、インターバルタイマのチェックは行なわれ、記憶されますので、INTERVAL ONが実行すると、INTERVAL STOP命令実行中にインターバルタイマが指定された時間を経過していた場合には、すぐに割り込みを発生させます。

サンプルプログラム

```
10 ON INTERVAL=120 GOSUB 50
20 TIME=0
30 INTERVAL ON
40 GOTO 40
50 BEEP
60 PRINT TIME
70 RETURN
```

(2秒ごとに、ブザーを鳴らし、時間を60倍の秒単位で表示します。)

ON KEY GOSUB

(オン キー ゴーサブ)

機能 ファンクションキーの割り込みが発生したときに飛ぶ、処理サブルーチンの開始行番号を指定します。

書式 ON KEY GOSUB <行番号> [, <行番号>...]

文例 ON KEY GOSUB100, 200, 300

(F1キーが押されたら行番号100へ、F2キーなら200へ、F3キーなら300へ処理が移るよう設定します。)

解説

この文が有効となるためには、以前にKEY(n)ON命令が実行される必要があります。処理ルーチンでは、RETURN文が実行されるまでファンクションキーの割り込みは受けつけません。処理ルーチンから復帰すると、再び割り込みを受けつけ

ようになります。

ベーシックプログラムを実行中でないと、割り込みは発生しません。またON ERROR GOTO文と同時に使用することはできません。

KEY (n) ON/OFF/STOP (キー(エヌ)オン/オフ/ストップ)

機能 指定されたファンクションキーによる割り込みの発生を許可、禁止、または停止します。

- 書式**
- ①KEY (<ファンクションキー番号>) ON
 - ②KEY (<ファンクションキー番号>) OFF
 - ③KEY (<ファンクションキー番号>) STOP

解説

① KEY(<ファンクションキー番号>)ON
ON KEY GOSUB<行番号>でファンクションキー番号による割り込みルーチンが指定されている場合に、新しい命令文を実行するたびに、ファンクションキー番号の状態をチェックし、押されると、割り込みが発生させます。

②KEY(<ファンクションキー番号>)OFF
割り込みの発生は禁止され、ファンクションキーの状態のチェックも行ないません。

③KEY (<ファンクションキー番号>)STOP
割り込みの発生は禁止されますが、ファンクションキーのチェックは行なわれ、その内容は記憶されています。

KEY(<ファンクションキー番号>)ONを実行すると、KEY (<ファンクションキー番号>)STOP 命令の実行中に、ファンクションキーが押されていた場合には、自動的に割り込みを発生させます。

サンプルプログラム

```
10 PRINT "F1, F2キーをおしてください"
20 ON KEY GOSUB 90, 100
30 KEY(1) ON
40 KEY(2) ON
50 GOTO 50
60 KEY(1) OFF
70 KEY(2) OFF
80 END
90 BEEP: RETURN 60
100 FOR I=1 TO 30: BEEP: NEXT I
110 RETURN 70
```

(ファンクションキー[F1]が押されるとブザーを1回だけ鳴らし、[F2]キーが押されるとブザーを30回鳴らして、実行を終了します。)

ON SPRITE GOSUB

(オン スプライト ゴーサブ)

機能 スプライトどうしの衝突があったとき、指定した処理ルーチンへ飛びます。

書式 ON SPRITE GOSUB <行番号>

文例 ON SPRITE GOSUB 1000
(スプライトどうしの衝突があったとき行番号1000へ飛びます。)

解説

スプライトどうしの衝突があったとき<行番号>で指定した割り込み処理ルーチンへサブルーチン・ジャンプします。

処理ルーチンの内部では、割り込みは受けつけられません。RETURN文が実行され、もとのルー

チンへ戻ってきたときに、再び割り込みを受けつけるようになります。

また、割り込みはプログラムを実行中でないと発生しません。ON ERROR GOTO文と同時に使用することはできません。

SPRITE ON/OFF/STOP

(スプライト オン/オフ/ストップ)

機能 スプライトの衝突による割り込みを制御します。

書式 ①SPRITE ON
②SPRITE OFF
③SPRITE STOP

解説

①SPRITE ON

スプライトの衝突による割り込みを可能にしたい場合、この命令を実行します。この命令を実行後、割り込みがあった場合、プログラムの流れは、ON SPRITE GOSUB文で指定した処理ルーチンへと移ります。

②SPRITE OFF

スプライトの衝突による割り込みが発生しないようにします。この命令を実行後は、次のSPRITE ON命令を実行するまではON SPRITE GOSUB文は何の意味も持ちません。

③SPRITE STOP

スプライトの衝突による割り込みを中止します。SPRITE OFFと違うのは、中止の間にスプライト

が衝突したか否かを覚えているということです。中止の間に衝突があった場合、次にSPRITE ON命令が実行されれば、すぐに割り込みが発生します。

サンプルプログラム

```
10 SCREEN 2,0
20 FOR I=1 TO 8
30 READ A: B$=B$+CHR$(A)
40 NEXT I
50 SPRITE$(0)=B$
60 ON SPRITE GOSUB 140
70 SPRITE ON
80 FOR I=0 TO 192
90 PUT SPRITE 1, (50+I, 10+I), 1, 0
```

```
100 PUT SPRITE 2, (220-1, 180-1), 8, 0
110 NEXT I
120 DATA 0, 16, 40, 124, 254, 84, 40, 0
130 END
140 SPRITE OFF
```

```
150 FOR J=1 TO 10: BEEP: NEXT J
160 RETURN 130
```

(スプライト面番号1と2のスプライトの衝突が発生すると、プログラムの実行が140行目に移り、ブザーを10回鳴らして実行を終了します。)

ON STOP GOSUB (オン ストップ ゴーサブ)

機能 **CTRL** + **STOP** キーが押されて、割り込みが発生したときに飛ぶサブルーチンの開始行を指定します。

書式 **ON STOP GOSUB**〈行番号〉

文例 **ON STOP GOSUB 200**
(**CTRL** + **STOP** キーが押されたら、行番号200の割り込み処理サブルーチンに飛びます。)

解説

CTRL + **STOP** キーが押されるたびに、〈行番号〉

で始まる割り込みサブルーチンを実行します。

この文を有効とするためには、キーが押される以前に、STOP ON命令を実行しておく必要があります。

割り込みルーチンでは、**CTRL** + **STOP** キーによる割り込みは起こりません。

ベーシック・プログラムの実行中にのみ有効です。

エラーなどにより、プログラムの実行が停止している場合には、無効となります。

ON ERROR GOTO文と同時に使用することはありません。

注)

ON STOP GOSUB 〈行番号〉には、プログラムの実行を停止する **CTRL** + **STOP** キーの操作が、命令の中に含まれますので、プログラムの実行が停止できなくなる可能性があります。次のようなプログラムを実行すると、電源を切る以外に方法はありません。

```
10 ON STOP GOSUB 40
20 STOP ON
30 GOTO 30
40 RETURN
```


STOP ON/OFF/STOP

(ストップ オン/オフ/ストップ)

機能 **CTRL** + **STOP** キーによる割り込みの発生を許可、禁止または停止します。

書式 ①STOP ON

②STOP OFF

③STOP STOP

解説

①STOP ON

新しい命令文を実行するたびに、**CTRL** + **STOP** キーの状態をチェックし、**CTRL** + **STOP** キーが押されると、ON STOP GOSUB<行番号>で指定した割り込みを発生させます。

②STOP OFF

割り込みの発生は禁止され、**CTRL** + **STOP** キーの状態のチェックを行いません。

③STOP STOP

割り込みの発生は禁止されますが、**CTRL** + **STOP** キーのチェックは行なわれ、記憶されますので、STOP ONを実行するとSTOP STOP命令実行中に **CTRL** + **STOP** キーが押されていた場

合には、自動的に割り込みを発生させます。

サンプルプログラム

```
10 ON STOP GOSUB 70
20 STOP ON:CLS
30 FOR I=0 TO 500
40 LOCATE 3, 10:PRINT"PU SH CTRL&STOP KEYS";
50 NEXT I
60 END
70 PRINT I
80 RETURN 60
```

(**CTRL** + **STOP** キーが押されると、そのときの I の値を表示してプログラムの実行を終了します。)

ON STRIG GOSUB

(オン エストリガ ゴーサブ)

機能 ジョイスティックのトリガボタンが押されて、割り込みが発生したときに飛ぶサブルーチンの開始行を指定します。

書式 ON STRIG GOSUB<行番号>[, <行番号>,]
<行番号> は最大 4 つ

文例 ON STRIG GOSUB 100, 200

解説

トリガボタンが押されるたびに、<行番号>で始まる割り込みサブルーチンを実行します。

この文を有効とするためには、トリガボタンが押される以前に STRIG ON 命令を実行しておく必要があります。

割り込みルーチンでは、トリガボタンによる割り込みは起こりません。

飛び先の行番号と、使用されるトリガボタンとの対応は、つぎのとおりです。

0 ……スペースバー

1, 3 ……ジョイスティック 1

2, 4 ……ジョイスティック 2

プログラムの実行中にのみ、この命令は有効です。

ON ERROR GOTO 文と同時に使用することはできません。

STRIG ON/OFF/STOP

(エストリガ オン/オフ/ストップ)

機能 ジョイスティックのトリガボタンによる割り込みを許可、禁止、または停止する。

- 書式**
- ①STRIG (〈ボタン番号〉) ON
 - ②STRIG (〈ボタン番号〉) OFF
 - ③STRIG (〈ボタン番号〉) STOP

解説

〈ボタン番号〉は使用されるトリガボタンの種類を指定します。〈ボタン番号〉の値は、0～4までで、使用されるトリガボタンとの対応はつぎのとおりです。

ボタン番号	使用されるトリガボタン
0	スペースバー（トリガボタンとして使用）
1 または 3	ジョイスティック 1 のトリガボタン
2 または 4	ジョイスティック 2 のトリガボタン

①STRIG 〈ボタン番号〉 ON

命令文が実行されるたびにトリガボタンが押されたかどうかをチェックします。トリガボタンが押されると、ON STRIG GOSUB 〈行番号〉による割り込みが発生させます。

②STRIG 〈ボタン番号〉 OFF

割り込みの発生を禁止し、トリガボタンが押されたかどうかのチェックも行ないません。

③STRIG 〈ボタン番号〉 STOP

割り込みは禁止されますが、トリガボタンの状態はチェックされ、記憶されます。STRIG (〈ボタン番号〉) STOP 命令の実行中にトリガボタンが押されていた場合には、STRIG (〈ボタン番号〉) ON 命令を実行すると、自動的に割り込みが発生させます。

サンプルプログラム

```

10 ON STRIG GOSUB 60
20 STRIG (0) ON
30 PRINT "PUSH! SPACE BAR"
40 GOTO 40
50 END
60 BEEP
70 RETURN 50

```

（スペースバーを押すと、ブザーを鳴らし、プログラムの実行を終了します。）

とく しゆ 特殊ステートメント

POKE (ポーク)

➡ 基礎編 179 ページ

機能 指定した番地のメモリにデータを書き込みます。

書式 POKE <メモリの番地>, <データ>

文例 POKE &HD000, 20
(メモリの&HD000番地に数値20を書き込みます。)

解説

<メモリの番地>は-32768~65535(&H0~&HFFF
F)、<データ>は0~255の範囲になければなりません。
いずれも小数部分は切り捨てられます。

<メモリの番地>が負の数の場合、65536を加えたものが実際の値になります。たとえばPOKE-1, 255
はPOKE 65535, 255と同じです。

なお番地&HF380以降は、ベーシックが使用して
いますので、指定できません。

VPOKE

(バイポーク)

機能 VRAM内の指定したアドレスにデータを書き込みます。

書式 VPOKE <VRAMのアドレス>, <データ>

文例 VPOKE &H1000, &H28
(VRAMアドレスの&H1000番地に、&H28を書き込みます。)

解説

VRAM(ビデオRAM)は、画面に表示される文字や図形を記憶しているメモリです。このメモリに書き込まれたデータがビデオ信号に変えられ、表示されます。

<VRAMのアドレス>で指定したVRAMに<データ>で指定した数値を書き込みます。<VRAMのアドレス>は0~16383(&H0~&H3FFF)、<データ>は0~255(&H0~&HFF)の範囲になければなりません。

注) VRAMのマッピング、各パターン・テーブルの位置(BASE命令を参照してください)などをよく理解した上で、この命令を使うようにしてください。

DEF USR

(デファイン ユーザ)

機能 機械語サブルーチンの開始番地を定義します。

書式 DEF USR [<番号>]=<開始番地>

文例 DEF USR 1=&HF000
(機械語ユーザ関数USR1の実行開始番地を&HF000に設定します。)

解説

機械語ユーザ関数USR0~USR9が呼び出す機械語サブルーチンの実行開始番地を設定します。<番号>は0~9で、省略すると「0」とみなします。

また、DEF USR文は2重に定義してもエラーにはなりません。後から定義した方が有効になります。

OUT

(アウト)

機能 出力ポートに1バイトのデータを送ります。

書式 OUT <ポート番号>, <式>

文例 OUT &HFF, 128
(CPUのI/O 出力ポート255番に数値128を出力します。)

解説

指定した<ポート番号>のCPUのI/O 出力ポートに<式>の値を出力します。<ポート番号>および<式>の値は0～255(&H0～&HFF)の範囲になれば

いけません。

ただし、<ポート番号>が255を超えても、エラーは出ません。
通常この命令を使うことはないでしょう。

WAIT

(ウェイト)

機能 指定した入力ポートから特定のデータを読み込む間、プログラムの実行を停止します。

書式 WAIT <ポート番号>, <式1> [, <式2>]

文例 WAIT 255, 1
(ポート番号255からのデータの最下位ビットが1になるまで実行を停止します。)

解説

<ポート番号>で指定したポートから読み込まれたデータを<式2>とEXCLUSIVE ORし、さらに<式1>とANDした結果が0の間プログラムの実行を停止し、ポートから読み込みを続けます。そして、0でなくなれば次の命令から実行を再開します。

<式1>、<式2>は値が整数(0～255)でなければいけません。

<式2>が省略された場合、その値は0とみなします。

CALL

(コール)

機能 ROMカートリッジで用意された拡張ステートメントを呼び出します。

書式 CALL <拡張ステートメント名>[(<式>, <式>, ……)]

解説

省略形として「_」(アンダーライン)を使用することができます。

この命令に関する詳しい説明は、各 ROM カートリッジの説明書をご覧ください。

MOTOR ON/OFF

(モータ オン/オフ)

機能 カセットテープレコーダのモータのオン/オフをコントロールします。

書式 ①MOTOR

②MOTOR ON

③MOTOR OFF

解説

①MOTOR

カセットレコーダのモータが現在動いていればオフに、止まっていれば、オンにします。

②MOTOR ON

カセットレコーダのモータはオンになり、動きま

す。

③MOTOR OFF

カセットレコーダのモータはオフになり、止まります。

数値関数

ABS関数 (アブソリュート)

➡ 基礎編 116 ページ

機能 絶対値を与えます。

書式 ABS (<引数>)

解説
<引数>の絶対値を与えます。

$ABS(X) \equiv |X|$

サンプルプログラム

```
PRINT ABS(-10) RETURN
```

(-10の絶対値を与えます。)

FIX関数 (フィックス)

機能 整数を与えます。

書式 FIX (<引数>)

解説

引数の整数部分を与えます。つまり $\text{SGN}(X) * \text{INT}(\text{ABS}(X))$ と同じになります。
INT関数とFIX関数との大きな違いは、引数が負の場合、FIX関数は単にその整数部分に「-」をつけたものを返すという点にあります。

サンプルプログラム

```
10 A = -3.14
20 PRINT FIX(A)
30 PRINT INT(A)
40 END
```

(FIX関数とINT関数の<引数>に-3.14を入れた結果を表示します。)

INT関数 (インテジャー)

→ 基礎編 117 ページ

機能 整数化を行ないます。

書式 INT (<引数>)

解説

結果は<引数>を超えない最大の整数となります。

$\text{INT}(X) \equiv [X]$

サンプルプログラム

```
PRINT INT (3.14) RETURN
```

(3.14の小数点部分を切り捨てて整数3を表示します。)

RND関数 (ランダム)

→ 基礎編 119 ページ

機能 0 から 1 の間の乱数を発生します。

書式 RND (<引数>)

解説

0 より大きく、1 よりも小さい乱数を発生します。
<引数>の値が正の場合は同一乱数系列の次の乱数を発生します。<引数>が0の場合は1つ前に発生した乱数と同じ数を与えます。<引数>が負の場合は引数の値によって決まる乱数を発生します。

サンプルプログラム

```
10 FOR I = 1 TO 5
20 A = RND (1)
30 PRINT A
40 NEXT I
50 END
```

(乱数を5個発生させ、画面に表示します。)

SGN関数^{かん すう}

(サイン)

→ 基礎編^{きそへん} 117 ページ

機能^{きのう} 符号^{ふごう}を与^{あた}えます。

書式^{しょしき} SGN (<引数>^{ひきすう})

解説^{かいせつ}

<引数>^{ひきすう}が正^{せい}ならば1を、<引数>^{ひきすう}が0ならば0を、<引数>^{ひきすう}が負^ふならば-1を^{あた}えます。

サンプルプログラム

PRINT SGN (-5) RETURN

(引数^{ひきすう}-5の符号^{ふごう}を与^{あた}えて、画面^{がめん}に出力^{しゅつりょく}します。)

SIN関数^{かん すう}

(サイン)

機能^{きのう} 正弦^{せいげん} (サイン) を与^{あた}えます。

書式^{しょしき} SIN (<引数>^{ひきすう})

解説^{かいせつ}

<引数>^{ひきすう}の正弦^{せいげん}を^{あた}えます。<引数>^{ひきすう}の単位^{たん い}はラジアンです。

$\text{SIN}(X) \equiv \sin X$

サンプルプログラム

PRINT SIN (3.14159/6) RETURN

(SIN $\pi/6$ を、画面^{がめん}に表示^{ひょうじ}します。)

COS関数^{かん すう} (コサイン)

機能^{きのう} 余弦^{よげん} (コサイン) ^{あた}を与えます。

書式^{しょしき} COS (<引数>^{ひきすう})

解説^{かいせつ}

<引数>^{ひきすう}の余弦^{よげん} ^{あた}を与えます。単位^{たんい}はラジアンです。

$\text{COS}(X) \equiv \cos X$

サンプルプログラム

PRINT COS (3.14159/3) RETURN

(COS $\pi/3$ を画面^{がめん}に表示^{ひょうじ}します。)

TAN関数^{かん すう} (タンジェント)

機能^{きのう} 正接^{せいせつ} (タンジェント) ^{あた}を与えます。

書式^{しょしき} TAN (<引数>^{ひきすう})

解説^{かいせつ}

<引数>^{ひきすう}の正接^{せいせつ} ^{あた}を与えます。<引数>^{ひきすう}の単位^{たんい}はラジアンです。

$\text{TAN}(X) \equiv \tan X$

サンプルプログラム

PRINT TAN (3.14159/4) RETURN

(TAN $\pi/4$ を画面^{がめん}に表示^{ひょうじ}します。)

ATN関数^{かん すう} (アーク タンジェント)

機能^{きのう} 逆正接^{ぎゃくせいせつ} (アークタンジェント) ^{あた}を与えます。

書式^{しょしき} ATN (<引数>^{ひきすう})

解説^{かいせつ}

<引数>^{ひきすう}の逆正接^{ぎゃくせいせつ}をラジアンで与^{あた}えます。逆正接^{ぎゃくせいせつ}は正接^{せいせつ} (tan タンジェント) の逆三角関数^{ぎゃくさんかくかんすう}です。与^{あた}えられる数値範囲^{すうちはんい}は、 $-\pi/2$ から $\pi/2$ までです。

$\text{ATN}(X) \equiv \tan^{-1} X$

サンプルプログラム

PRINT 4 * ATN (1) RETURN

($\text{TAN}^{-1}(1) * 4$ を画面^{がめん}に表示^{ひょうじ}します。)

SQR^{かん すう}関数

(スクエア ルート)

^{き のう}機能 ^{へいほうこん あた}平方根を与えます。

^{しょしき}書式 ^{ひきすう}SQR (<引数>)

^{かいせつ}解説

^{ひきすう}<引数>の平方根を与えます。^{ひきすう}<引数>は0 ^{いじょう}以上でなければなりません。

$$\text{SQR}(X) \equiv \sqrt{X}$$

サンプルプログラム

PRINT SQR (3)

($\sqrt{3}$ の^{あたい}値を画面に^{しやうりやく}出力します。)

EXP^{かん すう}関数

(エクスポネンシャル)

^{き のう}機能 ^{じょう あた}eのべき乗を与えます。

^{しょしき}書式 ^{ひきすう}EXP (<引数>)

^{かいせつ}解説

^{ひきすう}eの<引数>乗を与えます。eは^{し ぜんたいすう}自然対数の^{てい}底です。

$$\text{EXP}(X) \equiv e^X$$

^{ひきすう}引数の^{あたい}値は、145.06286085862 ^{い か}以下でなければなりません。

サンプルプログラム

PRINT EXP (1)

(eの^{あたい}値を画面に^{ひょうじ}表示します。)

LOG関数^{かん すう}

(ログ)

機能^{きののう} 自然対数を与えます。^{し ぜんたいすう あた}

書式^{しょしき} LOG (<引数>)^{ひきすう}

解説^{かいせつ}

<引数>^{ひきすう}の自然対数を与えます。<引数>^{ひきすう}は正の値でなければいけません。^{せい あたい}

LOG(X) \equiv loge X

サンプルプログラム

PRINT LOG (2) **RETURN**

(2の自然対数を画面に出力します。)^{し ぜんたいすう が めん しゅつりょく}

CINT関数^{かん すう}

(シー インテジャー)

機能^{きののう} 数値を整数型に変換します。^{すう ち せいすうがた へんかん}

書式^{しょしき} CINT (<引数>)^{ひきすう}

解説^{かいせつ}

<引数>^{ひきすう}の値を整数型に変換します。<引数>^{ひきすう}の値は-32768～32767の範囲になければなりません。^{あたい せいすうがた へんかん はん い}

なお、小数部分は切り捨てられます。^{しゅうすう ぶ ぶん き す}

サンプルプログラム

PRINT CINT (290.29) **RETURN**

(引数290.29を整数型に変えて画面に表示します。)^{ひきすう せいすうがた か が めん ひょうじ}

CDBL関数

(シー ダブル)

機能 数値を倍精度型に変換します。

書式 CDBL (<引数>)

解説

<引数>の値を倍精度型に変換します。単精度型、整数型の変数を精度よく計算したいときなどに用います。

サンプルプログラム

```
PRINT CDBL (1.23E+10) RETURN
```

(単精度型定数1.23E+10を倍精度型に変換して表示します。結果は12300000000となります。)

CSNG関数

(シー シングル)

機能 数値を単精度型に変換します。

書式 CSNG (<引数>)

解説

<引数>の値を7桁目で四捨五入して単精度型に変換します。メモリの節約にはなりますが、精度が落ちることに注意してください。

サンプルプログラム

```
PRINT CSNG (1.23456789) RETURN
```

(引数1.23456789を単精度型に変えて、画面に表示します。)

も　じ　かん　すう 文字関数

LEFT\$関数 (レフト ドル)

➡ 基礎編 109 ページ

機能 文字列の左側から指定した長さの文字列を与えます。

書式 LEFT\$ (〈文字式〉, 〈引数〉)

解説

〈文字式〉の左から〈引数〉の数だけの文字で構成される文字列を与えます。〈引数〉の範囲は0から255までで小数部分が切り捨てられてから関数が評価されます。〈引数〉が0の場合、関数は、「」(ヌルストリング)を与えます。また、〈引数〉が、〈文字式〉の文字数より大きければ、関数は〈文字式〉すべてを与えます。

サンプルプログラム

PRINT LEFT\$

("MICROCOMPUTER", 5) RETURN

(文字列 "MICROCOMPUTER" の左端から5文字目までの値を与えます。結果は "MICRO" となります。)

RIGHT\$関数 (ライト ドル)

→ 基礎編 110 ページ

機能 文字列の右側から指定した長さの文字列を与えます。

書式 RIGHT\$ (<文字式>, <引数>)

解説

<文字式>の右側から<引数>で指定した数の文字列を与えます。<引数>の範囲は0から255までで、小数部分が切り捨てられてから関数が評価されます。<引数>が0のとき、関数は「」(ヌルストリング)を与えます。また、<文字式>を構成する文字数より<引数>が大きい場合は、<文字式>すべてを与えます。

サンプルプログラム

PRINT RIGHT\$

("PERSONALCOMPUTER", 8) RETURN

(文字列"PERSONALCOMPUTER"中の右から8文字目までを与えます。結果は、"COMPUTER"となります。)

MID\$関数 (ミドル ドル)

→ 基礎編 110 ページ

機能 文字列の中から指定した長さの文字列を抜き出します。

書式 MID\$ (<文字式>, <引数 1> [, <引数 2>])

解説

<文字式>において、左から<引数 1>番目の文字から、<引数 2>で指定した文字数の文字列を抜き出し与えます。<引数 1>は1~255、<引数 2>は0~255の範囲で指定します。また小数部分を切り捨ててから関数を評価します。<引数 2>が0の場合および<文字式>の長さよりも大きい場合は、「」(ヌルストリング)を与えます。<引数 2>を省略した場合、および<文字式>の<引数 1>文字目以降の長さよりも<引数 2>の方が大きい場合は、<引数 1>文字目以降のすべてを与えます。

サンプルプログラム

PRINT MID\$

("You have a computer." 5, 4) RETURN

(文字列"You have a computer."中の、左から5番目のhから4文字を抜き出します。結果は、"have"となります。)

LEN関数 (レングス)

→ 基礎編 111 ページ

機能 文字列の文字数を与えます。

書式 LEN (<文字式>)

解説

<文字式>を構成している文字の数を与えます。このとき、画面に出力されない文字（スペースやキャラクターコードの1から31まで）も数えます。

サンプルプログラム

```
PRINT LEN ("MICROCOMPUTER") RETURN
```

(文字列 "MICROCOMPUTER" の文字数を与えます。結果は、13となります。)

ASC関数 (アスキー)

→ 基礎編 156 ページ

機能 キャラクターコードを与えます。

書式 ASC (<文字式>)

解説

<文字式>の先頭の文字のキャラクターコードを与えます。キャラクターコードと文字の対応は、「キャラクターコード表」を参照してください。

注) <文字式>が、「" "」(ヌルストリング)のときには、Illegal function call エラーになります。

サンプルプログラム

```
PRINT ASC("Y")
```

(Yのキャラクターコードを画面に表示します。)

CHR\$関数 (キャラクター ドル)

→ 基礎編 154 ページ

機能 キャラクターコードに対応する文字を与えます。

書式 CHR\$ (<引数>)

解説

<引数>をキャラクターコードとする文字を与えます。<引数>の範囲は0から255までで、小数部分が切り捨てられてから、関数が評価されます。キャラクターコードについては、「キャラクターコード表」を参照してください。

サンプルプログラム

```
PRINT CHR$(&H4F)
```

(キャラクターコード&H4Fに対応する文字を画面に表示します。)

VAL関数 (バル)

→ 基礎編 151 ページ

機能 文字列を数値に変換します。

書式 VAL (<文字式>)

解説

<文字式>の表す数値を与えます。<文字式>の最初の文字が「+」、「-」、「&」、数字のいずれでもなければ0を与えます。数字（16進数の場合はA～Fの英字も含む。先頭に&Hをつけます。）以外の文字が現われると、それ以降の文字はすべて無視されます。<文字式>中の空白はすべて無視されます。

サンプルプログラム

```
10 A$="12":B$="34"
20 C$=A$+B$
30 C=VAL(A$)+VAL(B$)
40 PRINT C,C$
50 END
```

(文字列12と34を、行番号20でつなぎ、数値12+34の値を行番号30で計算して画面に表示します。)

STR\$関数 (ストリング ドル)

→ 基礎編 152 ページ

機能 数値を文字列（数字）に変換します。

書式 STR\$ (<引数>)

解説

<引数>の数値を表わす文字列を与えます。

サンプルプログラム

```
10 A=12:B=34
20 C=A+B
30 C$=STR$(A)+STR$(B)
40 PRINT C,C$
50 END
```

(数値12+34の値と、文字列12と34をつないだ値を画面に表示します。)

STRING\$関数

(ストリング ドル)

機能 1文字を指定した数だけ繰り返した文字列を与える。

書式 ① STRING\$ (<繰り返しの数>, <キャラクターコード>)
② STRING\$ (<繰り返しの数>, <文字式>)

解説

- ①の場合、キャラクターコードに対応する文字列を<繰り返しの数>だけ連ねた文字列を与えます。
②の場合、<文字式>の値の先頭文字を<繰り返しの数>だけ連ねた文字列を与えます。
<繰り返しの数>、<キャラクターコード>はともに

0～255の範囲になければいけません。ただし、電源投入時、<繰り返しの数>は、0～200の範囲で、これを変更する場合には、CLEAR 命令を用います。小数部分は切り捨てられます。

サンプルプログラム

```
PRINT STRING$ (5, "*") RETURN
```

(*を5個表示します。)

SPACE\$関数

(スペース ドル)

機能 指定した数だけ空白を与えます。

書式 SPACE\$ (<引数>)

解説

<引数>個の空白からなる文字列を与えます。<引数>の値は0から255まで許されます。
ただし、電源投入時は、0～200の範囲で、これを変更する場合には、CLEAR 命令を用います。小数部分は切り捨てられます。

サンプルプログラム

```
PRINT SPACE$ (5); "K" RETURN
```

(画面左から5文字分の空白の後、Kを表示します。)

INSTR^{かん すう}関数 (インストリング)

機能 文字列を検索します。

書式 INSTR ([<引数>], <文字式 1>, <文字式 2>)

解説

<文字式 1>の中の<引数>文字目 (省略すると1文字目) 以降からの<文字式 2>を検索し、先頭文字から何番目にあるかを与えます。<引数>は0~255の範囲で指定でき、小数点以下があれば切り捨てられます。

<引数>が<文字式 1>の長さより大きい場合や、<文字式 1>が「 ” 」(ヌルストリング)の場合、または<文字式 2>が<文字式 1>の中に見つからない場合は0を与えます。<文字式 2>がヌルストリングの場合、<引数>(省略したときは1)の値をそのまま与えます。

サンプルプログラム

```
10 B=0:A$="CENTIMETER"  
20 B=INSTR(B+1,A$,"E")  
30 PRINT B;  
40 IF B>0 THEN 20  
50 END
```

(文字変数A\$="CENTIMETER"の中に、"E"という文字が先頭(B+1)から何番目にあるかをさがして画面に表示します。)

INKEY\$^{かん すう}関数 (インキー ドル)

➡ 基礎編125ページ

機能 キーが押されていれば、その文字を、キーが押されていなければ、「 ” 」(ヌルストリング)を与えます。

書式 INKEY\$

解説

[CTRL] + [STOP] 以外のキーが押されていれば、その文字を、キーが押されていなければ、「 ” 」(ヌルストリング)を与えます。キーが押されるまで、待つ場合にはINPUT\$関数が便利です。

サンプルプログラム

```
10 A$=INKEY$  
20 IF A$="" THEN GOTO 10  
30 PRINT A$  
40 END
```

(キーボードから押された文字を画面に表示します。)

INPUT\$関数

(インプット ドル)

機能 キーボードから指定された文字数の文字列を入力します。

書式 INPUT\$ (<引数>)

解説

キーボードから<引数>で指定した文字数の文字列を入力します。キーボードから入力されるキャラクターは、プログラムを中断する **CTRL** + **STOP** を除いて全て読み込まれます。

<引数>の値は1～255まで指定できます。ただし、201以上の値を使用するときにはCLEAR命令で文字領域の大きさを設定しておく必要があります。

また、引数の小数部分は切り捨てられます。

サンプルプログラム

```
10 A$=INPUT$(7)
```

```
20 PRINT "HITACHI"+A$
```

```
30 END
```

(キーボードから適当な文字を7文字入力すると、HITACHIのすぐ後に続いて、入力した文字が表示されます。)

BIN\$関数

(バイナリー ドル)

機能 数値を2進数に変えた結果を文字列で与えます。

書式 BIN\$ (<引数>)

解説

<引数>を2進数で表す文字列に変換します。<引数>は-32768～65535の範囲になければなりません。<引数>が負の場合、それに65536を加えたものが、実際の引数となります。

また、小数部分は切り捨てられます。

サンプルプログラム

```
PRINT BIN$ (6305) RETURN
```

(6305を2進数に変換して表示します。)

OCT\$関数^{かん すう}

(オクト ドル)

機能 数値を8進数に変えた結果を文字列で与えます。

書式 OCT\$ (<引数>)

解説
引数を8進数で表わす文字列に変換します。<引数>
は-32768~65535の間で指定します。
<引数>が負の場合、それに65536を加えたものが、
実際の引数となります。

サンプルプログラム

```
PRINT OCT$ (56) RETURN
```

(10進数(56)を、8進数に変えて表示します。結果
は70になります。)

HEX\$関数^{かん すう}

(ヘキサ ドル)

機能 数値を16進数に変えた結果を文字列で与えます。

書式 HEX\$ (<引数>)

解説
<引数>を16進数で表わす文字列に変換します。<引
数>は-32768~65535の間の値で指定します。
<引数>が負の場合、それに65536を加えたものが、
実際の引数となります。

サンプルプログラム

```
PRINT HEX$ (255) RETURN
```

(10進数(255)を16進数に変えて表示します。)

とく しゅ かん すう 特殊関数

かん すう PEEK関数 (ピーク)

→ 基礎編 179 ページ

機能 メモリ番地の内容を与えます。

書式 PEEK (<メモリ番地>)

解説

<メモリ番地>の内容 1 バイトを与えます。<メモリ番地>は式の形で指定でき、範囲は、-32768~65536 で小数部分が切り捨てられた後、実行されます。指定した番地が負の数の場合には、65536を加えたものが実際の値になります。

サンプルプログラム

PRINT PEEK (256) RETURN
(メモリ番地256の内容を表示します。)

かん すう VPEEK関数 (バイ ピーク)

機能 VRAM内の指定したアドレスの内容を与えます。

書式 VPEEK (<アドレス>)

解説

<アドレス>で指定したVRAMの内容 1 バイト (0 ~ 255) を与えます。<アドレス>は 0 ~ 16383 (&H0 ~ &H3FFF) の範囲になければなりません。

サンプルプログラム

```
PRINT VPEEK (&H1000) RETURN
(VRAMアドレス&H1000番地の内容を表示します。)
```

USR関数 (ユーザ)

機能 ユーザが定義した機械語サブルーチンを呼び出します。

書式 USR [<番号>] (<引数>)

解説

<引数>を機械語サブルーチンに渡して、サブルーチンを呼び出します。<番号>は 0 から 9 ままで、DEFUSR文で定義した番号に対応します。また、<番号>を省略すると「0」になります。

ベーシックから機械語への引数の引き渡しは、メモリを用いて行ないます。引数の型に従って以下の 4 つの場合があります。

- i) 整数型 &HF7F8、&HF7F9番地に引数が格納されます。&HF7F8番地に下位バイトがはいるので注意してください。また&HF663番地に、整数型を示す数値「2」が格納されます。
- ii) 単精度型 &HF7F6 ~ &HF7F9番地の 4 バイトに引数が格納されます。&HF663番地には「4」がはいらいます。
- iii) 倍精度型 &HF7F6 ~ &HF7FD番地の 8 バイトに引数が格納されます。&HF663番地には「8」がはいらいます。
- iv) 文字型 &HF7F8 ~ &HF7F9番地の 2 バイト

に、ストリング・ディスクリプタの番地が格納されます。ストリング・ディスクリプタは 3 バイトよりなり、先頭は文字の長さを示し、以下の 2 バイトで実際に文字のある番地を示しています。

&HF663番地には「3」が入ります。

VARPTR関数

(バー ポインター)

機能 変数のデータが格納されている先頭番地を与えます。

書式 ① VARPTR (<変数名>)
② VARPTR (#<ファイル番号>)

解説

①では変数や配列の要素のデータの格納されている先頭番地を与えます。<変数名>は変数名または配列の要素で指定します。

VARPTR関数実行の前に、その変数には、値が代入されていなければなりません。

またVARPTRの返す値は、-32768～32767です。
負の数のアドレスが返された場合、それに65536を加えた値が、実際のアドレスとなります。

②では、指定されたファイルを制御する部分の先頭番地を与えます。

サンプルプログラム

```
10 A=3
20 ADRS=VARPTR(A)
30 PRINT "a="; ADRS
40 END
```

(変数Aのデータが格納されている番地を画面に表示します。)

INP

(インプット)

機能 入力ポートから1バイトのデータを読み込みます。

書式 INP (<ポート番号>)

解説

指定した<ポート番号>の入力ポートから1バイトのデータを読み込みます。<ポート番号>の値は0～255(&H0～&HFF)の範囲でなければいけません。

サンプルプログラム

```
PRINT INP (&HFF) RETURN
```

(ポート番号255番から1バイトのデータを読み込みます。)

TAB関数

(タブ)

→ 基礎編 101 ページ

機能 指定した水平位置までカーソルを移動します。

書式 TAB (<引数>)

解説

PRINTまたは、LPRINT文中で用いられ、<引数>で指定される水平位置までスペースを出力しながらカーソルを移動します。<引数>は0以上255以下の数値で、与えた<引数>が画面1行の表示文字数を超えていたら、1行の表示文字数で引いた値が用いられます。

サンプルプログラム

```
PRINT TAB (10); "A" RETURN
```

(画面水平位置10までカーソルを移動させます)

SPC関数

(スペース)

→ 基礎編 100 ページ

機能 指定された個数の空白を出力します。

書式 SPC (<引数>)

解説

引数の数の空白を出力します。PRINT、LPRINT、PRINT # 文の中でのみ使用することができます。

引数の値は0～255の間で指定します。

サンプルプログラム

```
PRINT SPC (5); "A" RETURN
```

(空白を5つ画面に出力します)

FRE関数 (フリー)

機能 未使用領域を与えます。

書式 FRE (<引数>)

解説

① FRE(0)

<引数>が数式の場合はメモリの未使用領域の大きさをバイト数で与えます。

② FRE(" ")

<引数>が文字式の場合にはメモリの文字領域のうち、使っていない領域の大きさをバイト数で与えます。

サンプルプログラム

```
10 CLEAR 300, &HE000
20 PRINT FRE(0); FRE(" ")
30 END
```

(メモリ番地の上限を&HE000、文字領域の大きさを300バイトに設定した後、メモリの未使用領域並びに、未使用の文字領域の大きさを画面に表示します。)

EOF関数 (エンド オブ ファイル)

機能 ファイルが終了したかどうかを判定します。

書式 EOF (<ファイル番号>)

解説

<ファイル番号>で指定した入力ファイルが終了している場合は、-1を、そうでなければ0を与えます。<ファイル番号>はOPEN命令ですでに開かれていなければなりません。

サンプルプログラム

```
10 OPEN "CAS:TEST" FOR OUTPUT AS #1
20 PRINT #1, 1;2;3;4;5;6;7;8;9;0
30 CLOSE
40 END
50 MAXFILES=2
60 OPEN "CAS:TEST" FOR INPUT AS #1
70 OPEN "CRT:" FOR OUTPUT AS #2
80 INPUT #1, A$
90 PRINT #2, A$
100 IF EOF(1)=-1 THEN PRINT EOF(1)
110 CLOSE:END
```

(行番号10~40:カセットレコーダに1~0までの数値データを記録します。行番号50~110:カセットレコーダから行番号40までのプログラムで記録したデータを読み込み、変数A\$に代入し、画面に表示します。ファイルが終了すると、-1を表示します。)

ERL関数

(エラー ライン)

機能 エラーの発生した行番号を与えます。

書式 ERL

解説

エラーが発生したときの行番号を与えます。コマンド待ちの状態エラーが発生したときは、65535を与えます。エラー回復処理ルーチンで、どの行でエラーが発生したかを調べるのに便利です。

注)IF文でERL関数の値を調べるとき、行番号は等号の右に書いてください。左に書くと、RENUMコマンドを実行したとき行番号が修正されません。

サンプルプログラム

```
10 ON ERROR GOTO 60
20 A=25
30 B$="YOKO"
40 PLINT A, B$
50 END
60 IF ERR=2 THEN PRINT ERL
70 RESUME 50
```

(エラー(この場合、Syntax error)が発生した行を画面に表示します。)

ERR関数

(エラー)

機能 発生したエラーのエラーコードを与えます。

書式 ERR

解説

エラーが発生したときのエラーコードを与えます。エラー回復処理ルーチンで、どのようなエラーが発生したかを調べるのに便利です。

エラーコードとその意味は、「エラーメッセージ一覧表」を参照してください。

CSRLIN^{かん すう}関数

(カーソル ライン)

^{き のう}機能 画面上のカーソルの^{すいちよく い ち}垂直位置^{あた}を与えます。

^{しょしき}書式 CSRLIN

^{かいせつ}解説

^{げんざい}現在のカーソルの^{すちよく い ち}垂直位置を0～22(KEY OFF で
0～23)の^{ぎょうたん い}行単位^{あた}で与えます。^{ひきすう}引数はありません。

サンプルプログラム

10 CLS

20 LOCATE 1, 7

30 PRINT CSRLIN

40 END

(カーソルの^{すいちよく い ち}垂直位置(この場合は7)を^{がめん ひょう}画面に表
^じ示します。)

LPOS関数 (エル ポジション)

機能 プリントヘッドの現在位置を与えます。

書式 LPOS (<引数>)

解説

プリンタバッファにおけるプリントヘッドの現在の水平方向の位置を与えます。

<引数>は意味を持ちませんが、1文字記入が必要です。

サンプルプログラム

```
10 LPRINT "ATOSYUAZMUAKI";  
20 PRINT "げんざいのヘッドのいちは"; LPOS  
    (0)  
30 END
```

(行番号10の文字列をプリンタに印字した後、プリントヘッドの位置を画面に表示します)

PAD関数 (パッド)

機能 タッチパッドの状態を返します。

書式 PAD (<引数>)

解説

指定した<引数>の値により、次のようなタッチパッドの状態を返します。

引数の値	タッチパッドの接続されたジョイスティックポート	返される値の内容	返される値
0	1	タッチパッドに触れている 触れていない	- 1 0
1	1	タッチパッドのX座標	座標値
2	1	タッチパッドのY座標	座標値
3	1	タッチパッドのスイッチが 押されている 押されていない	- 1 0
4	2	タッチパッドに触れている 触れていない	- 1 0
5	2	タッチパッドのX座標	座標値
6	2	タッチパッドのY座標	座標値
7	2	タッチパッドのスイッチが 押されている 押されていない	- 1 0

PAD(0) または PAD(4) が評価されたときにはじめて、そのタッチパッドの座標値が有効になりますので、2個のタッチパッドを同時に接続して使用した場合、ジョイスティック1に接続したタッチパッドの状態が、ジョイスティック2に接続したタッチパッドの座標値に悪影響を及ぼすことがあります。

そのため、2つのタッチパッドの値を連続して読むときには、2つのPAD関数の間に、何行かのプログラムを入れるか、待ちループを入れます。

PDL関数 (パドル)

機能 パドルの値を返します。

書式 PDL (<引数>)

解説

<引数>は、1 から12の範囲で、パドルの接続されたポートを指定します。<引数>が奇数のときは、ジョイスティック 1 に接続されたパドル、偶数の

ときは、ジョイスティック 2 に接続されたパドルの値を返します。

PLAY関数 (プレイ)

機能 指定したPSGチャンネルがPLAY命令を実行中であるか否かを数値で返します。

書式 PLAY (<チャンネル番号>)

文例 X=PLAY(3):PRINT X

解説

<チャンネル番号>で指定したチャンネルが、PLAY命令を実行中である場合には、-1、そうでない場合には、0を返します。

<チャンネル番号>は、0～3の範囲で指定し、0が指定されると、全てのチャンネルについてチェックし、どれか1つでも実行中の場合には、-1を返します。

注) PLAY命令のすぐ後に、PLAY関数を入れると、現在の状態にかかわらず、-1を返すことがあります。

サンプルプログラム

```
10 PLAY "CDEFGAB"  
20 X=PLAY (0)  
30 PRINT X  
40 END
```

(行番号10で音を発生させ、PLAY命令を実行中であることを画面に表示します。)

POINT関数^{かん すう}

(ポイント)

機能 ^{き のう} グラフィック座標^{ざひょう}の点^{てん}のカラーコード^{しら}を調べます。

書式 ^{しょしき} POINT (〈水平位置〉^{すいへい い ち}, 〈垂直位置〉^{すいちよく い ち})

解説 ^{かいせつ}
画面上^{が めんじょう}の指定^{ししてい}した位置^{い ち}のカラーコードを 0 ～15 で
あた^{あた}えます。

サンプルプログラム

```
10 SCREEN 2
20 PSET (200, 90), 1
30 PSET (100, 10), 8
40 A=POINT (200, 90)
50 B=POINT (100, 10)
60 SCREEN 1
70 PRINT A, B
80 END
```

(グラフィック画面^{が めん}に行番号^{ぎょうばんごう}20, 30で打たれた
点^{てん}の色^{いろ}を、テキスト画面^{が めん}にカラーコードで表示^{ひょうし}
します。)

POS関数^{かん すう}

(ポジション)

機能 ^{き のう} 画面上^{が めんじょう}のカーソルの水平位置^{すいへい い ち}の値^{あたい}をあた^{あた}えます。

書式 ^{しょしき} POS (〈引数〉^{ひきすう})

解説 ^{かいせつ}
〈引数〉^{ひきすう}を「0」にすると、画面上^{が めんじょう}でのカーソルの水
平位置^{すいへい い ち}をあた^{あた}えます。POS(0)が持つ値^{あたい}の範囲^{はんい}は、
0 ～28です。(SCREEN 0を指定^{ししてい}すると、0 ～38)
〈引数〉^{ひきすう}は意味^{い み}を持ちませんが、1文字^{も じ}記入^{きにゅう}が必要^{ひつよう}
です。

サンプルプログラム

```
10 CLS
20 LOCATE 20, 20
30 PRINT POS(0)
40 END
```

(カーソルを20, 20の位置^{い ち}に移動^{いどう}させた後^{あと}、カー
ソルの水平位置^{すいへい い ち}を画面^{が めん}に表示^{ひょうし}します。)

STICK関数^{かん すう}

(スティック)

機能 指定したカーソルキーまたは、ジョイスティックの方向を0～8の数値で返します。

書式 STICK (〈ジョイスティック番号〉)

解説

〈ジョイスティック番号〉は0から2の数値で、使用するジョイスティック、またはカーソルコントロールキーを指定します。

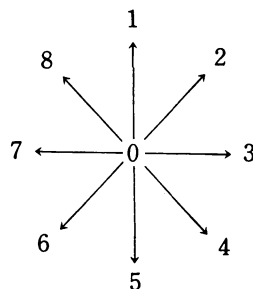
ジョイスティック番号	使用するジョイスティック
0	カーソルコントロールキー(をジョイスティックとして使用する場合)
1	ジョイスティック 1
2	ジョイスティック 2

ジョイスティックまたは、カーソルコントロールキーの方向により、次の値を返します。

サンプルプログラム

```
10 PRINT "カーソルコントロールキーをおして  
    ください"  
20 X=STICK(0)  
30 PRINT X  
40 GOTO 20  
50 END
```

(押されたカーソルコントロールキーの方向を画面に、数値で表示します。)



STRIG関数^{かん すう}

(エス トリガ)

機能 ジョイスティックのトリガボタンの状態を返します。

書式 STRIG (〈ボタン番号〉)

解説

〈ボタン番号〉で指定したジョイスティックのトリガボタンの状態を返します。

トリガボタンが押されると、-1を返します。それ以外では0を返します。

サンプルプログラム

```
10 PRINT "スペースバーをおしてください"  
20 X=STRIG(0)  
30 PRINT X  
40 GOTO 20  
50 END
```

(スペースバーを押すと-1、それ以外のときは0を画面に表示します。)

とく しゆ へん すう 特殊変数

TIME (タイム)

➡ 基礎編 157 ページ

機能 内蔵クロックのセットおよびクロックの 1/60 秒ごとのカウント数を与えます。

書式 ① TIME
② TIME = <式>

解説

① TIME

内蔵クロックのカウント数を与えます。値は、1/60秒単位です。ただし割り込みを用いる処理（一連の割り込み関係の命令や、カセットへの入出力）によって、多少の時間のずれを生じます。

② TIME = <式>

内蔵クロックを<式>の値にセットしなおします。

<式>の値は0～65535まで使用できます。

サンプルプログラム

```
10 TIME=0:PRINT TIME
20 FOR N=1 TO 100
30 NEXT N
40 PRINT TIME
50 GOTO 20
60 END
```

(経過した時間を0から画面に表示します。)

SPRITE\$

(スプライト ドル)

→ 基礎編 146 ページ

機能 スプライトのパターンを作ります。

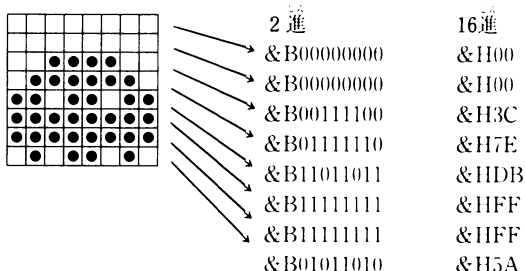
書式 SPRITE\$ (<パターン番号>)=<文字式>

解説

<文字式>で表わされるパターンを <パターン番号>で指定したスプライトに定義します。

<パターン番号>は、SCREEN命令で指定したスプライトサイズが0、1の場合は0から255まで、スプライトサイズが2、3の場合は0から63まで指定できます。

<文字式>は、スプライト上のドット(点)を表わす2進数をキャラクターコードとする文字列を指定します。下図のような図形をパターン番号1のスプライトとする場合

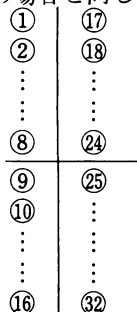


SPRITE\$(1)=CHR\$(&H00)+CHR\$(&H00)+CHR\$(&H3C)+CHR\$(&H7E)+CHR\$(&HDB)+CHR\$(&HFF)+CHR\$(&HFF)+CHR\$(&H5A)

と指定します。

8×8スプライトでは8文字ぶん、16×16スプライトでは32文字ぶんの文字列を指定しなければなりません。それに満たない部分は0で満たされます。

16×16スプライトでは、下図のような順で文字列を指定します。ドット指定は、8ドット×8ドットの場合と同じです。



VDP

(バイディーピー)

機能 VDP (ビデオ信号IC)のレジスタの値を設定します。

書式 VDP (<引数>)

解説

<引数>の値が、0～7の範囲では、VDPに8つある書き込み専用レジスタの値を設定します。<引数>が8のときは、VDPの、読み取り専用のステータスレジスタの値を与えます。

通常、この命令を使うことはないでしょう。

なお、この特殊変数VDP、ならびにBASEを用いる方はこれらの変数の持つ働きを正確に理解してからお使いください。

BASE

(ベース)

機能 VDPレジスタの各テーブルの先頭番地を設定します。

書式 BASE (<引数>)

解説
 <引数>で指定される VDP レジスタ内の各テーブルの先頭番地を設定します。
 <引数>と各テーブルの対応はつぎのとおりです。

引数	テ ー ブ ル
0	テキストモードのパターン名称テーブルの先頭番地
1	意味を持ちません
2	テキストモードのパターンジェネレーターテーブルの先頭番地
3	意味を持ちません
4	意味を持ちません
5	テキストモードのパターン名称テーブルの先頭番地
6	テキストモードのカラーテーブルの先頭番地
7	テキストモードのパターンジェネレーターテーブルの先頭番地
8	テキストモードのスプライト属性テーブルの先頭番地
9	テキストモードのスプライトパターンテーブルの先頭番地
10	ハイリゾリューションモードのパターン

引数	テ ー ブ ル
	名称テーブルの先頭番地
11	ハイリゾリューションモードのカラーテーブルの先頭番地
12	ハイリゾリューションモードのパターンジェネレーターテーブルの先頭番地
13	ハイリゾリューションモードのスプライト属性テーブルの先頭番地
14	ハイリゾリューションモードのスプライトパターンテーブルの先頭番地
15	マルチカラーモードのパターン名称テーブルの先頭番地
16	意味を持ちません
17	マルチカラーモードのパターンジェネレーターテーブルの先頭番地
18	マルチカラーモードのスプライト属性テーブルの先頭番地
19	マルチカラーモードのスプライトパターンテーブルの先頭番地

上記のうち、0～4までは、39×24行テキストモード、5～9までは、29×24行テキストモードのそれぞれの先頭番地です。

また、10～14が、ハイリゾリューションモード、15～19が、マルチカラーモードのそれぞれの先頭番地です。

さくいん 索引

コマンド、ステートメント アルファベット順^{じゅん}

AUTO(オート)	190
BEEP(ビープ)	228
BLOAD(ビーロード)	196
BSAVE(ビーセーブ)	196
CALL(コール)	248
CIRCLE(サークル)	225
CLEAR(クリア)	198
CLOAD(シーロード)	194
CLOAD?(シーロードベリファイ)	195
CLOSE(クローズ)	215
CLS(クリア スクリーン)	220
COLOR(カラー)	221
CONT(コンティニュー)	193
CSAVE(シーセーブ)	194
DATA(データ)	209
DEF FN(デファイン ファンクション)	211
DEF INT/SNG/DBL/STR(デファイン インテジャー/シングル/ダブル/ストリング)	211
DEF USR(デファイン ユーザ)	246
DELETE(デリート)	191
DIM(ディメンジョン)	210
DRAW(ドロー)	227
END(エンド)	204
ERASE(イレース)	210
ERROR(エラー)	235
FOR~NEXT(フォー~ネクスト)	200
GOSUB~RETURN(ゴースブ~リターン)	201
GOTO(ゴートウー)	201

IF~THEN~ELSE(イフ~ゼン~エルス).....	202
INPUT(インプット).....	204
INPUT#(インプット シャープ).....	217
INPUT\$(インプット ドル).....	219
INTERVAL ON/OFF/STOP(インターバル オン/オフ/ストップ).....	239
KEY(キー).....	233
KEY ON/OFF (キー オン/オフ).....	234
KEY ON/OFF/STOP(キー オン/オフ/ストップ).....	240
KEY LIST(キーリスト).....	234
LET(レット).....	199
LINE(ライン).....	224
LINE INPUT(ライン インプット).....	205
LINE INPUT#(ライン インプット シャープ).....	218
LIST(リスト).....	189
LLIST(エルリスト).....	190
LOAD(ロード).....	195
LOCATE(ロケイト).....	220
LPRINT(エルプリント).....	207
LPRINT USING(エルプリント ユージング).....	208
MAXFILES(マックスファイルズ).....	214
MERGE(マージ).....	197
MID\$(ミドル ドル).....	212
MOTOR ON/OFF (モータ オン/オフ).....	248
NEW(ニュー).....	192
ON ERROR GOTO(オン エラー ゴートウー).....	236
ON GOTO/ GOSUB(オン ゴートウー/ ゴーサブ).....	203
ON INTERVAL GOSUB(オン インターバル ゴーサブ).....	238
ON KEY GOSUB(オン キー ゴーサブ).....	239
ON SPRITE GOSUB (オン スプライト ゴーサブ).....	241
ON STOP GOSUB(オン ストップ ゴーサブ).....	242
ON STRIG GOSUB(オン エストリガ ゴーサブ).....	243
OPEN(オープン).....	215
OUT(アウト).....	247
PAINT(ペイント).....	225
PLAY(プレイ).....	228

POKE(ポーク)	245
PRESET(ピーリセット)	224
PRINT(プリント)	205
PRINT USING(プリント ユージング)	206
PRINT # (プリント シャープ)	216
PRINT # USING (プリント シャープ ユージング)	216
PSET(ピーセット)	223
PUT SPRITE(プット スプライト)	226
READ(リード)	208
REM(リマーク)	200
RENUM(リナンバー)	191
RESTORE(リストア)	209
RESUM (リジューム)	236
RUN(ラン)	192
SAVE(セーブ)	196
SCREEN(スクリーン)	222
SOUND(サウンド)	230
SPRITE ON/OFF/STOP(スプライト オン/オフ/ストップ)	241
STOP(ストップ)	203
STOP ON/OFF/STOP(ストップ オン/オフ/ストップ)	243
STRIG ON/OFF/STOP(エストリガ オン/オフ/ストップ)	244
SWAP(スワップ)	213
TRON/OFF(トレース オン/オフ)	193
VPOKE(バイポーク)	246
WAIT(ウェイト)	247
WIDTH(ウィドス)	221

索引

かんすう とく しゅ へん すう じゅん
関数、特殊変数 アルファベット順

ABS(アブソリュート).....	249
ASC(アスキー).....	258
ATN(アークタンジェント).....	252
BASE(ベース).....	277
BIN\$(バイナリー ドル).....	262
CDBL(シーダブル).....	255
CHR\$(キャラクター ドル).....	258
CINT(シー インテジャー).....	254
COS(コサイン).....	252
CSNG(シー シングル).....	255
CSRLIN(カーソル ライン).....	270
EOF(エンド オブ ファイル).....	268
ERL(エラーライン).....	269
ERR(エラー).....	269
EXP(エクスポネンシャル).....	253
FIX(フィックス).....	249
FRE(フリー).....	268
HEX\$(ヘキサ ドル).....	263
INKEY\$(インキー ドル).....	261
INP(インプット).....	266
INPUT\$(インプット ドル).....	262
INSTR(インストリング).....	261
INT(インテジャー).....	250
LEFT\$(レフト ドル).....	256
LEN(レングス).....	258
LOG(ログ).....	254
LPOS(エル ポジション).....	271
MID\$(ミドル ドル).....	257

OCT\$(オクトドル).....	263
PAD(パッド).....	271
PDL(パドル).....	272
PEEK(ピーク).....	264
PLAY(プレイ).....	272
POINT(ポイント).....	273
POS(ポジション).....	273
RIGHT\$(ライトドル).....	257
RND(ランダム).....	250
SGN(サイン).....	251
SIN(サイン).....	251
SPACE\$(スペースドル).....	260
SPC(スペース).....	267
SPRITE\$(スプライトドル).....	276
SQR(スクエアルート).....	253
STICK(スティック).....	274
STR\$(ストリングドル).....	259
STRIG(エストリガ).....	274
STRING\$(ストリングドル).....	260
TAB(タブ).....	267
TAN(タンジェント).....	252
TIME(タイム).....	275
USR(ユーザ).....	265
VAL(バル).....	259
VARPTR(バーポインター).....	266
VDP(ブイデーピー).....	276
VPEEK(ブイピーク).....	264

ふろく 付録

100

100

サンプルプログラム

1 カラー調整

```
10 ON STOP GOSUB 260
20 COLOR 15,1
30 SCREEN 2
40 STOP ON
50 OPEN "GRP:"AS #1
60 FOR Y=1 TO 3
70 FOR X=1 TO 5
80 C=(Y-1)*5+X
90 CIRCLE(40*X,48*Y+Z),15,C
100 IF X=1 AND Y=1 THEN CIRCLE(40,48),15
110 PAINT(40*X,48*Y+Z),C
120 NEXT X
130 Z=Z+10
140 NEXT Y
150 DRAW"BM38,20"
160 PRINT#1,"1____2____3____4____5"
170 DRAW"BM38,78"
180 PRINT#1,"6____7____8____9____10"
190 DRAW"BM36,135"
200 PRINT#1,"11____12____13____14____15"
210 DRAW"BM85,5"
220 PRINT #1,"カラー_ちょうせい"
230 CLOSE
240 GOTO 240
250 END
260 COLOR 15,4,7
270 STOP OFF:RETURN 250
```



15色の円が描かれ、その上にカラーコードが表示
されます。テレビの色調整つまみなどで、正しい
色になるように画面の色を調節してください。
終わるときは、**CTRL** + **STOP** キーです。

2 カラーデモ1

```
10 SCREEN 2
20 FOR Y=0 TO 127 STEP 10
30 FOR X=0 TO 192 STEP 8
40 LINE(X,Y)-(255-X,191-Y),C,BF
50 C=C+1
60 IF C=15 THEN C=1
70 NEXT X,Y
80 GOTO 20
90 END
```

色紙を重ねるように、カラーコードの1から15までの色を次々に表示します。画面中央までくると今度は1枚1枚を貼がしていきます。重ねたり貼がしたりするたびに色紙は小さくなっていきます。

スピード切換スイッチをスロー2、スロー3にすると、ゆっくり変化します。また、**[STOP]** キーを押すと変化が止まり、もう一度 **[STOP]** キーを押すとまた変化します。

終わるときは **[CTRL] + [STOP]** キーです。

3 カラーデモ2

```
10 ON STOP GOSUB 220
20 COLOR 15,1
30 SCREEN 2
40 STOP ON
50 LINE(125,0)-(125,191)
60 LINE(0,95)-(255,95)
70 FOR Y=0 TO 40 STEP .25
80 PX=255*RND(1):PY=191*RND(1)
90 C=15*RND(1)
100 PSET(PX,PY),C
110 NEXT Y
120 FOR Y=0 TO 58 STEP 2
130 R=50*SIN(2*Y*3.14/191)
140 C=15*RND(1)
150 CIRCLE(125,Y),R,C,...4
160 CIRCLE(125,191-Y),R,C,...4
170 CIRCLE(19+Y,95),R,C,...5
180 CIRCLE(231-Y,95),R,C,...5
190 NEXT Y
200 GOTO 200
210 END
220 COLOR 15,4,7
230 STOP OFF:RETURN 210
```

4つに区切られた画面に、星のような点があられ、その後4つのだ円が色を変えながら描かれます。

終わるときは **[CTRL] + [STOP]** キーです。

4 カラーデモ3

```
10 ON STOP GOSUB 390
20 COLOR 15,1:T=0
30 SCREEN 2
40 STOP ON
50 C1=15*RND(SIN(T))
60 IF C1=1 OR C1=0 THEN GOTO 50
70 PX=120*RND(1):PY=160*RND(1):R=1
80 X=PX+80:Y=PY
90 C=C1:GOSUB 340
100 C=C1:GOSUB 360
110 PX=PX+8
120 R=R+1
130 IF R<=10 GOTO 90
140 PX=PX-80:R=1
150 C=1:GOSUB 340
160 C=C1:GOSUB 360
170 PX=PX+8
180 R=R+1
190 IF R<=10 GOTO 150
200 PX=PX+8
210 C=1:GOSUB 360
220 C=C1:GOSUB 340
230 PX=PX+8
240 R=R-1
250 IF R>=1 THEN GOTO 210
260 PX=PX-88:R=10
270 C=1:GOSUB 340
280 C=1:GOSUB 360
290 PX=PX+8
300 R=R-1
310 IF R>=0 GOTO 270
320 T=T+1:IF T>1000 THEN T=0
330 GOTO 50
340 CIRCLE(PX,PY),5,C,,,7
350 RETURN
360 CIRCLE(X,Y),4*R,C,,,2
370 RETURN
380 END
390 COLOR 15,4,7
400 STOP OFF:RETURN 380
```

大きくなったり小さくなったりするだ円の中を、
くさが^{とお}り抜けてゆきます。[CTRL] + [STOP]
キーが押されるまで^おど^{なが}とん描かれます。

5 ユーフォー ちゃくりく UFOの着陸

```

10 COLOR 15,4,7
20 ON STOP GOSUB 720
30 SCREEN 2,2
40 STOP ON
50 LINE(0,120)-(255,191),2,BF
60 CIRCLE(30,150),30,15,,,5
70 LINE(60,150)-(0,150)
80 X=X+.5
90 IF X)=200 THEN X=0
100 LINE(45,150-7.5*SQR(3))-(15,150+7.5*SQR(3))
110 COLOR 6
120 DRAW"BM60,160R30M70,180R40M130,160R200"
130 DRAW"BM60,160M80,140R30M120,130R40M150,140R200"
140 PAINT(120,150),6
150 LINE(75,145)-(255,145),15
160 LINE(70,155)-(255,155),15
170 LINE(130,130)-(80,180),15
180 LINE(150,130)-(100,180),15
190 CIRCLE(172,180),6,6,,,5
200 PAINT(172,180),6
210 LINE(172,130)-(172,180),15
220 COLOR 5
230 DRAW"BM140,125M+20,+2R7E3M+10,-4M+7,+12M+20,-5R50"
240 DRAW"BM140,125M+12,-20E30R10F10R5M+10,-30E10R5F15E40"
250 PAINT(180,100)
260 COLOR 12
270 DRAW"BM100,120M-12,+8L10M-8,+2M-7,-2H7L5M-8,+5L2M-20,
    -5M-20,+8M-10,-3L20"
280 C=12
290 DRAW"BM100,120M-12,-6H25L10G4L5M-15,-20H10L5M-8,+5L30"
300 PAINT(60,100),C
310 FOR J=1 TO 11
320 B$=""
330 READ A$
340 FOR I=1 TO 32
350 B$=B$+CHR$(VAL("&H"+MID$(A$,2*I-1,2)))
360 NEXT I
370 SPRITE$(J)=B$
380 NEXT J
390 FOR K=0 TO 1
400 PUT SPRITE 1+K*10,(80+80*K,50-30*K),15,1
410 PUT SPRITE 2+K*10,(80+80*K,66-30*K),15,2
420 PUT SPRITE 3+K*10,(96+80*K,50-30*K),15,3
430 PUT SPRITE 4+K*10,(96+80*K,66-30*K),15,4
440 NEXT K
450 PUT SPRITE 5,(170,118),1,11

```


6 スプライトパターン^{さくせい}作成プログラム

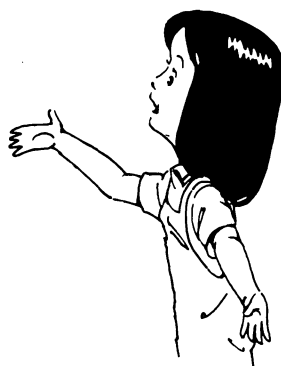
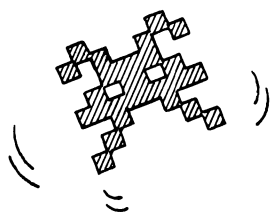
```
10 ON STOP GOSUB 500
20 STOP ON
30 CLS
40 PRINT"_____(パターン さくせい)"
50 PRINT
60 PRINT"____0123456789ABCDEF"
70 PRINT
80 FOR I=0 TO 15
90 PRINT"_:":HEX$(I):"....."
100 NEXT I
110 LOCATE 20,6:PRINT"_:ひょうは"
120 LOCATE 20,7:PRINT"_:カーソルキーで"
130 LOCATE 20,9:PRINT"_:セットは"
140 LOCATE 20,10:PRINT"_:Sキーで"
150 LOCATE 20,12:PRINT"_:リセットは"
160 LOCATE 20,13:PRINT"_:Rキーで"
170 LOCATE 20,15:PRINT"_:エントリは"
180 LOCATE 20,16:PRINT"_:Eキーで"
190 PX=0:PY=0
200 A$=INKEY$
210 LOCATE PX+4,3:PRINT"◆"
220 LOCATE 3,PY+4:PRINT"◆"
230 IF A$=CHR$(28) AND PX<15 THEN GOSUB 510:PX=PX+1
240 IF A$=CHR$(29) AND PX>0 THEN GOSUB 510:PX=PX-1
250 IF A$=CHR$(31) AND PY<15 THEN GOSUB 510:PY=PY+1
260 IF A$=CHR$(30) AND PY>0 THEN GOSUB 510:PY=PY-1
270 IF A$="s" OR A$="S" THEN LOCATE PX+4,PY+4:PRINT"●"
280 IF A$="r" OR A$="R" THEN LOCATE PX+4,PY+4:PRINT"."
290 IF A$="e" OR A$="E" THEN GOTO 320
300 GOTO 200
310 B=0
320 LOCATE PX+4,3:PRINT"_":LOCATE 3,PY+4:PRINT"_":FOR J=0 TO 15
330 A=0
340 FOR I=B TO B+7
350 A=A*2
360 P$=CHR$(VPEEK(6144+32*(J+4)+I+6))
370 IF P$="●" THEN A=A+1
380 NEXT I
390 Q$=RIGHT$("00"+HEX$(A),2)
400 K$=K$+Q$
410 NEXT J
420 B=B+8:IF B=8 THEN GOTO 320 ELSE 430
430 CLS
440 FOR I=0 TO 15
450 X$(1)=MID$(K$,2*I+1,2)
460 X$(2)=MID$(K$,33+2*I,2)
470 PRINT X$(1),X$(2)
480 NEXT I
```

```

490 END
500 CLS:STOP OFF:RETURN 490
510 LOCATE PX+4,3:PRINT"_"
520 LOCATE 3,PY+4:PRINT"_"
530 RETURN

```

たて16個、よこ16個の「・」を表示します。カーソルコントロールキー(␣␣␣␣)を使ってたてとよこの位置を決めて、[S]キーを押します。「・」は白丸になりますから、これをくり返して作りたいスプライトパターンを描いていきます。[S]キーを押し間違えたり、訂正したいときには、[R]キーを押します。スプライトパターンを作り終わったら、[E]キーを押します。しばらくすると画面には、16×16ドットのスプライトパターンを16進数に変換した値で表示します。149ページのように、左上の数字から文字列に変換していけばよいのです。左上の1/4の8×8個でパターンを作成すれば、8×8ドットのスプライトも作れます。



7 おいかっこ

```
10 KEY OFF
20 ON STOP GOSUB 500
30 STOP ON
40 DEFINT A-Z
50 MX=28:MY=23:DX=1:DY=1:NN=2:DL=500
60 XX=INT(RND(1)*MX):YY=INT(RND(1)*MY)
70 X=INT(RND(1)*MX):Y=INT(RND(1)*MY)
80 CLS:LOCATE 0,23:PRINT"SCORE":SC:
90 TIME=0
100 DR=INT(RND(1)*8):LOCATE XX,YY:PRINT"_":
110 IF DR=0 THEN DX=-DX:GOTO 140
120 IF DR=1 THEN DY=-DY:GOTO 140
130 IF DR=2 THEN DX=-DX:DY=-DY
140 XX=XX+DX:YY=YY+DY
150 IF XX>=MX THEN XX=MX-1
160 IF YY<0 THEN YY=0
170 IF YY>=MY THEN YY=MY-1
180 IF XX<0 THEN XX=0
190 LOCATE XX,YY:PRINT"@":
200 FOR II=1 TO DL:NEXT II
210 FOR JJ=1 TO NN:LOCATE X,Y:PRINT"_":
220 N=STICK(1)
230 IF 1=N THEN Y=Y-ABS(DY):GOTO 360
240 IF 7=N THEN X=X-ABS(DX):GOTO 360
250 IF 5=N THEN Y=Y+ABS(DY):GOTO 360
260 IF 3=N THEN X=X+ABS(DX):GOTO 360
270 IF 2=N THEN X=X+ABS(DX):Y=Y-ABS(DY):GOTO 360
280 IF 4=N THEN X=X+ABS(DX):Y=Y+ABS(DY):GOTO 360
290 IF 6=N THEN X=X-ABS(DX):Y=Y+ABS(DY):GOTO 360
300 IF 8=N THEN X=X-ABS(DX):Y=Y-ABS(DY):GOTO 360
310 N$=INKEY$
320 IF N$=CHR$(&H1E) THEN Y=Y-ABS(DY):GOTO 360
330 IF N$=CHR$(&H1D) THEN X=X-ABS(DX):GOTO 360
340 IF N$=CHR$(&H1F) THEN Y=Y+ABS(DY):GOTO 360
350 IF N$=CHR$(&H1C) THEN X=X+ABS(DX)
360 IF X=MX THEN X=MX-1:GOTO 400
370 IF X<0 THEN X=0:GOTO 400
380 IF Y=MY THEN Y=MY-1:GOTO 400
390 IF Y<0 THEN Y=0
400 LOCATE X,Y:PRINT"♥"
410 IF (X=XX) AND (Y=YY) THEN GOTO 440
420 NEXT JJ
430 GOTO 100
440 LOCATE 15,10:PRINT"CATCH"
450 S=TIME
460 SC=10000/S+SC:DL=DL-20
470 FOR I=1 TO 500:NEXT I
480 GOTO 60
```

490 END
500 CLS:KEY ON
510 STOP OFF:RETURN 490

画面には、「@」(アットマーク)と「♥」(ハートマーク)が出ます。あなたは、♥マークです。カーソルコントロールキー (⏏⏏⏏⏏) を使って、@マークを捕まえてください。(2つのキーを同時に押すと斜め方向に動きます。) うまくつかまえると、「CATCH」と表示します。スコアは時間制ですから、早くつかまえればつかまえるほどよい得点になります。

ジョイスティック(別売)をお持ちの方は、ジョイスティック接続コネクタ1に接続すれば使えます(斜め方向も使えます)。

終了するときは、**CTRL** + **STOP** キーを押します。



8 インベーダーダンス

```
10 COLOR 15,4,7
20 SCREEN2,0
30 I=1
40 SPRITE$(1)=CHR$(&H24)+CHR$(&H7E)+CHR$(&H99)+CHR$(&H99)+
  CHR$(&H99)+CHR$(&H7E)+CHR$(&H24)+CHR$(&HC3)
50 LINE(0,96+I)-(255,96+I)
60 LINE(0,96-I)-(255,96-I)
70 LINE(127,95)-(255-I,191)
80 LINE(127,95)-(I,0)
90 LINE(127,95)-(I,191)
100 LINE(127,95)-(255-I,0)
110 LINE(127,95)-(255,95+I)
120 LINE(127,95)-(255,95-I)
130 LINE(127,95)-(0,95+I)
140 LINE(127,95)-(0,95-I)
150 I=I*1.4
160 IF I<115 THEN GOTO 50
170 R=R+1
180 CIRCLE(127,95),R,1,,,2
190 IF R<80 GOTO 170
200 LINE(0,76)-(255,115),1,BF
210 X=X+6+9*RND(1):Y=10*SIN(.5*X)
220 PUT SPRITE 3,(X,Y+95),3,1
230 IF X>255 THEN X=0
240 GOTO 210
250 END
```



奥行きを感じさせる画面の中央を、インベーダー
が跳びはねます。終了させたいときは、**CTRL**
+ **STOP** キーを押します。

MB-H1のモニタ機能仕様

MB-H1の仕様

1. 概要

(1)モニタの起動

現在のモード	モニタの起動のしかた
メニュー画面	ファンクションキー[F2]を押す
ベーシック	Call mon 注1) または_mon

注1) 大文字、小文字のいずれでもかまいません。

(2)モニタコマンド一覧

コマンド 記号	コマンドフォーマット	機能
B	B add	ブレークポイントの設定
C	C add1, add2, add3	メモリ内容の比較
D	D add	メモリ内容の表示
E	E	モニタから抜け出て、元のモードに復帰
F	F add1, add2, data	指定のメモリブロックに定数を記録
G	G add	指定した番地からプログラムを実行
H	H	コマンドの機能の表示
I	I port	指定の I/O ポートの入力データの表示
M	M add	指定した番地のメモリ内容の表示と変更
O	O port, data	指定の I/O ポートへ出力
R	R reg	CPUレジスタの内容表示と変更
S	S	プログラムカウンタ(PC)の示すアドレスから1命令実行
T	T add1, add2, add3	指定したメモリブロックを他のアドレス領域に転送

reg:レジスタ add:メモリのアドレス port:I/O ポートアドレス data:1バイトデータ

2. コマンドのパラメータ

(1) コマンド待ち状態

「-」(マイナス記号)をラインの左端に表示し、カーソルはその右隣りにあります。

(2) ? 表示

正しくないコマンドフォーマットを指定したとき「?」を表示し、コマンド待ちとなります。

(3) メモリアドレス

- a) 16進数で入力、表示をします。(0~9、A~F)
- b) 4桁まで有効(0~FFFF)
- c) 5桁以上入力した場合、最後の4桁のみ有効となります。

例

FF 0 0 1 1 は 0 0 1 1 と入力したのと同じです。

(4) I/O ポートアドレス、データ

- a) 16進数で入力、表示をします。
- b) 2桁まで有効(0~FF)
- c) 3桁以上入力した場合、最後の2桁のみ有効となります。

例

0 1 F 0 は F 0 と入力したのと同じです。

(5) RETURN

- a) コマンドを実行します。(コマンドのフォーマットが間違っているときは「?」を表示します。)
- b) M RETURN 実行後、アドレスとデータを表示しているときにこのキーを押すとコマンド待ちに戻ります。

(6) , (カンマ)

アドレス、データ等コマンドの引数の区切り記号です。

(7) ␣ (スペース)

- a) Mコマンド実行中、アドレスとデータを表示しているときにスペースバーを押すと次のアドレスとデータを表示します。
- b) Mコマンド実行中以外ときには無視されます。(但し、カーソルは移動します。)

例

アドレス03␣2A は032A と同じです。

(8) / (スラント)

- a) Mコマンド実行中でアドレスとデータを表示しているとき、一つ前のアドレスのデータを表示します。
- b) Mコマンド以外ときには、現在のコマンドから抜け出し、コマンド待ちに戻ります。アドレスやデータなどの入力中に/を入力することにより、それをキャンセルできます。

注) メモリ内容を変更できるのはRAM領域に限ります。ROM領域の内容は変わりません。

3. コマンドの仕様

ブレイク

B (Break)

B add

1. 機能

ブレイクポイントの設定および変更を行います。

2. 操作

- B をキー入力すると、現在設定されているブレイクポイントを表示します。
- ブレイクポイントの変更をしないときは **RETURN** キーを押します。これによりコマンド待ちに戻ります。
- ブレイクポイントを変更するときは、a. の後、アドレスを表示しているときに、新しいアドレスを入力し **RETURN** キーを押します。これにより、ブレイクポイントはそのアドレスに設定され、コマンド待ちに戻ります。

注) ブレイクポイントの初期値には、0 番地が設定されています。また、ブレイクポイントがROM領域に設定されているときは、このブレイクポイントは無効になります。

コンペア

C (Compare)

C add1, add2, add3

1. 機能

- add 1 ~ add 2 と add 3 ~ add 3 + (add 2 ~ add 1) の2つのメモリブロックの内容を比較し、一致すれば、そのまま一致しなければ、「Compare error」を表示し、コマンド待ちに戻ります。

2. 操作

- C. add 1, add 2, add 3 **RETURN** とキー入力するとC コマンドを実行しコマンド待ちに戻ります。

注) add 1 > add 2 のときは実行せず「?」を表示してコマンド待ちに戻ります。

ディスプレイ

D (Display)

D add

1. 機能

add から64バイト分のメモリ内容を表示します。

2. 操作

- D をキー入力すると、前回D コマンドを実行したアドレスの次のアドレスを表示します。(前回の指

定アドレス+64が表示するアドレスとなります。)

b. 次に **RETURN** キーを押すと現在表示中のアドレスから64バイト分表示し、コマンド待ちとなります。

c. .a. のあと、新しいアドレスを設定し **RETURN** キーを押すと、新しく設定したアドレスから64バイト分表示し、コマンド待ちとなります。

注) D コマンドの add の初期値には、0 番地が設定されています。

エスケイプ

E (Escape)

E

1. 機能

a. モニタから抜け出て、元のモードに戻ります。

(メニュー画面からモニタになっていた場合はメニュー画面に、
ベーシックからモニタになっていた場合は、ベーシックに復帰します。)

2. 操作

a. E **RETURN** で元のモードに復帰します。

フィル

F (Fill)

F add1, add2, data

1. 機能

a. add1 ~ add2 のメモリブロックを data で満たします。

2. 操作

a. F add1, add2, data **RETURN** で実行し、コマンド待ちとなります。

注) add1 > add2 のときは実行せず「？」を表示します。

ゴー

G (Go)

G add

1. 機能

a. add からプログラムを実行します。

2. 操作

a. G をキー入力すると、プログラムカウンタ (PC) の値を表示します。

- b. プログラムのスタートアドレスが表示のアドレスのままであれば **RETURN** キーを押して、プログラムを実行します。
- c. スタートアドレスの変更が必要なときは、a. の後、新しいアドレスを入力し、**RETURN** キーを押すと、新しく入力したアドレスからプログラムを実行します。
- 注) 一度、G コマンドを実行しますと、プログラムカウンタ (PC) のアドレスから、プログラムを実行し続けます。これを停止させるのは、次の4つの方法によります。
- (1) ブレークポイントによる方法 (RAM 領域に設定) … B コマンド参照
 - (2) メインプログラム (ネスティング 0 の状態) で RET (機械語コード: C9) を実行する。
 - (3) 電源を切る。内部のプログラムは消えます。プログラムが暴走したときは、この方法だけが有効となります。
 - (4) JP 0 (0 番地へジャンプ: 機械語コードは、C3 0 0 0 0 の3バイト) あるいは、RST 0 (リスタート命令: 機械語コードは、C7) を実行したとき、この後、メニュー画面となりますが、メモリの内容は、だいたい保存されています。(一部、こわされている部分もあります。)

ヘルプ

H (Help)

H

1. 機能

コマンドの機能を表示します。

2. 操作

H **RETURN** とキー入力すると、下記のコマンドリストを表示し、コマンド待ちとなります。

B : Break point	I : Input I/O port
C : Compare	M : Memory
D : Display	O : Output I/O port
E : Escape	R : Register
F : Fill	S : Step
G : Go	T : Transfer
H : Help	

インプット

I (Input)

I port

1. 機能

指定の I/O ポートの入力データを表示します。

2. 操作

I port **RETURN** と入力すると入力データを表示し、コマンド待ちとなります。

M (Memory)

M add

1. 機能

指定番地のメモリ内容の表示および変更をします。

2. 操作

- M をキー入力すると前回の M コマンドで最後に表示したアドレスを表示します。
- そのアドレスでよければ **RETURN** キーを、あるいは変更の必要があれば新しいアドレスを入力後 **RETURN** キーを押すと
アドレス - データ
の形式で表示をします。
- データ部分を変更するときは、新しいデータを入力後スペースバーまたは「/」を入力します。
- スペースバーを押すと次のアドレスとデータを表示します。
- 「/」を入力すると1つ前のアドレスとデータを表示します。
- RETURN** キーを入力するとコマンド待ちに戻ります。このとき、最後に表示されていたアドレスのデータは以前のものが保持されます。

注)

- M コマンドの add の初期値には、0 番地が設定されています。
- システムワークエリア、ベーシックワークエリア（メモリ、アドレスは F0 0 0 ~ FFFF 番地）のデータを書きかえると正しい動作をしないことがありますのでこの領域は使用しないでください。

O (Output)

O port, data

1. 機能

指定の I/O ポートに 1 バイト data を出力します。

2. 操作

O port, data **RETURN** と入力すると、data を指定の I/O ポートに出力しコマンド待ちとなります。

注) システム I/O エリアにデータを書き込むと正しい動作をしなくなることがありますので注意が必要です。

R (Register)

R [reg]

1. 機能

CPUのレジスタ内容の表示および変更を行います。

2. 操作

- R **RETURN** とキー入力すると、IR レジスタ以外の CPU レジスタ内容を表示し、コマンド待ちに戻ります。
- R reg **RETURN** とキー入力すると指定のレジスタの内容を4桁(16進数)で表示します。
 - 変更の必要のない場合は **RETURN** キーを押します。これでコマンド待ちに戻ります。
 - 変更するときは、4桁以内の16進数(もし5桁以上入力した場合には最後の4桁のみ有効)を入力し、**RETURN** キーを押すとレジスタの内容を変更し、コマンド待ちに戻ります。
- reg に指定する文字(1文字または2文字)

reg	CPUのレジスタ
AまたはAF	AF
BまたはBC	BC
DまたはDE	DE
HまたはHL	HL
XまたはIX	IX
YまたはIY	IY
PまたはPC	PC
IまたはIR	IR
S	SP

reg	CPUのレジスタ
A'またはAF'	AF'
B'またはBC'	BC'
D'またはDE'	DE'
H'またはHL'	HL'

- reg は **RETURN** とキー入力される前の最後の1文字または2文字が有効
- reg に上記以外の文字を指定すると無視される(a.と同じ動作をする)かまたは「?」を表示してコマンド待ちに戻ります。

S (Step)

1. 機能

- プログラムをプログラムカウンタ(PC)の示すアドレスから1命令実行し、実行後のIRレジスタ以外のCPUレジスタ内容を表示し、コマンド待ちに戻ります。
- ブレークポイントが設定されていても、1命令トレースを実行します。

2. 操作

まず、R コマンドでプログラムカウンタ(PC)の内容を、S コマンドを実行するプログラムの先頭(RAM領域)に設定します。

b. コマンド待ちの状態^{まじょうたい}で S RETURN とキー入力^{にゅうりよく}すると、プログラムカウンタ (PC) のアドレスから 1 命令^{めいれい}トレース^{じっこう}を実行^{じっこう}し、実行後^{じっこうご}の IR レジスタ^{い がい}以外の CPU レジスタの内容^{ないよう}を表示^{ひょうじ}してコマンド待ち^{まち}に戻^{もど}ります。

注) プログラムカウンタ (PC) が ROM の領域^{りょういき}にある^{ある}ときには、S コマンドは正しく機能^{きんのう}しません。

トランスファー

T (Transfer)

T add1, add2, add3

1. 機能^{きののう}

add1 ~ add2 のメモリブロックの内容^{ないよう}を add3 ~ add3 + (add2 - add1) のメモリブロックへ転送^{てんそう}します。

2. 操作^{そうさ}

T add1, add2, add3 RETURN と入力^{にゅうりよく}するとブロック間転送^{かんてんそう}を実行^{じっこう}し、コマンド待ち^{まち}に戻^{もど}ります。

注) add1 > add2 のときは、実行^{じっこう}せず「?」を表示^{ひょうじ}してコマンド待ち^{まち}に戻^{もど}ります。

VDPについて

●H1 に実装されている VDP (ビデオ・ディスプレイ・プロセッサ、TMS9918A) は多くの機能を持った画面制御用 LSI で、以下に示すような特徴を持っています。

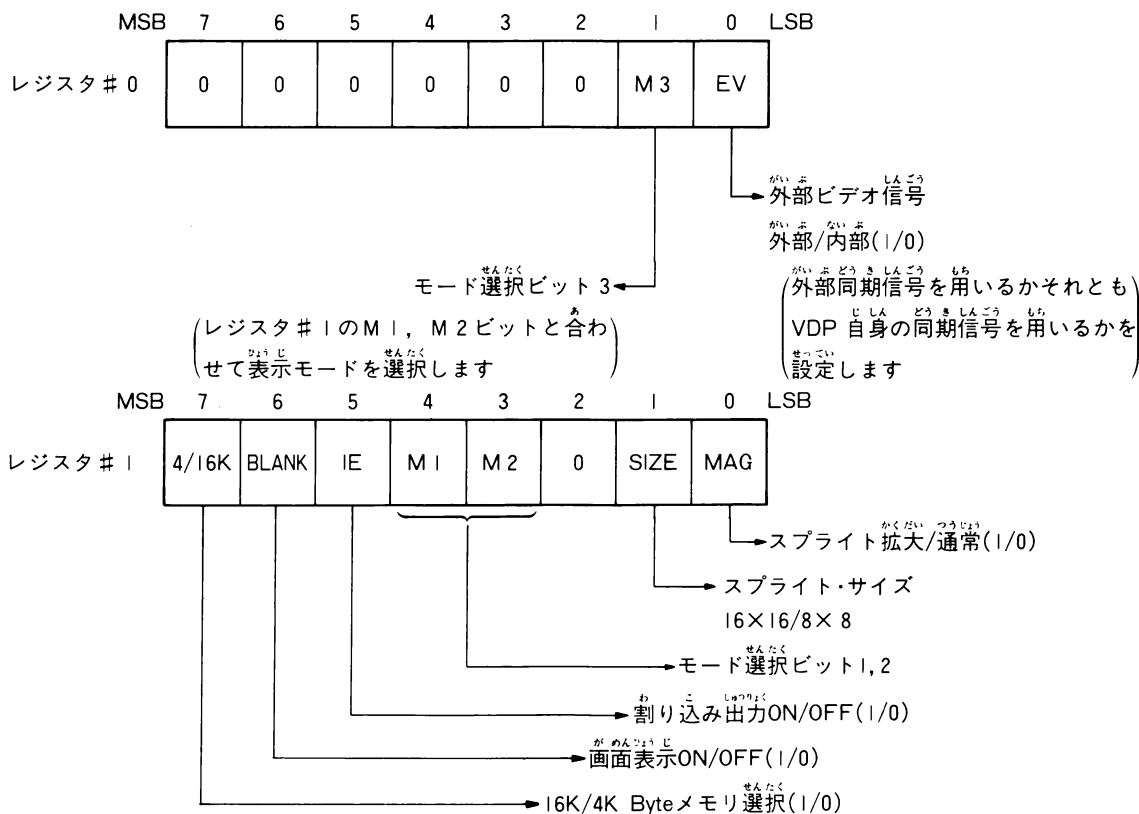
- i) 256ドット×192ドットのグラフィック表示
- ii) 32枚のスプライト面
- iii) 16色の同時表示
- iv) 1フレーム (1/60秒) ごとの割り込み発生機能
- v) ダイナミック RAM の自動リフレッシュ機能
- vi) NTSC 方式のコンポジット・カラー信号出力

VDP は8つの書き込み専用レジスタ (レジスタ#0～レジスタ#7) と読み出し専用レジスタ (ステータス・レジスタ) を持っています。これらのレジスタにデータを書き込むことによって、VDP の動作や V-RAM のマッピングを設定できます。またステータス・レジスタを読み出すことで、現在のVDPの動作を知ることができます。

各レジスタの機能を以下に示します。

i) レジスタ#0, レジスタ#1

VDP の機能および表示モードを設定します。



VDP の表示モードおよびその設定方法を次の表に示します。

VDP の表示モード

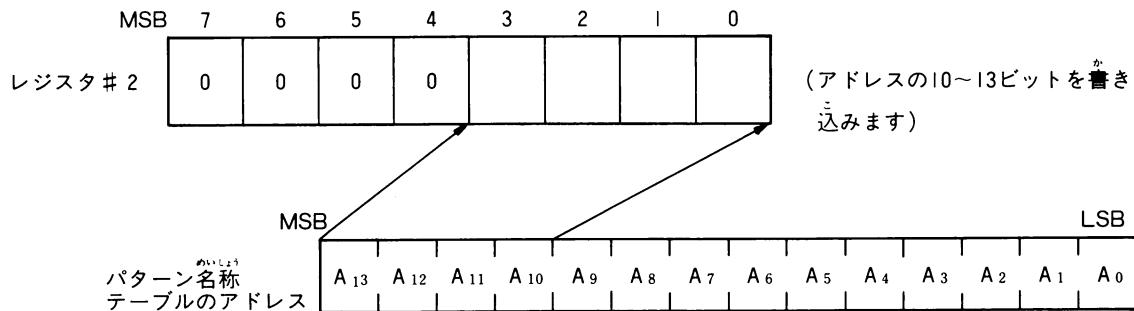
モード	解像度	パターン数	色指定	スプライト (動画)	BASIC モード
グラフィック I	192×256画素	256種類	16色	使用可	SCREEN 1
グラフィック II	192×256画素	768種類	16色	使用可	2
マルチカラー	48×64画素	—	16色	使用可	3
テキスト	24行×40列	256種類	16色のうち2色	使用不可	0

表示モードの設定

M 1	M 2	M 3	表示モード
0	0	0	グラフィック I
0	0	1	グラフィック II
0	1	0	マルチカラー
1	0	0	テキスト

ii) レジスタ# 2

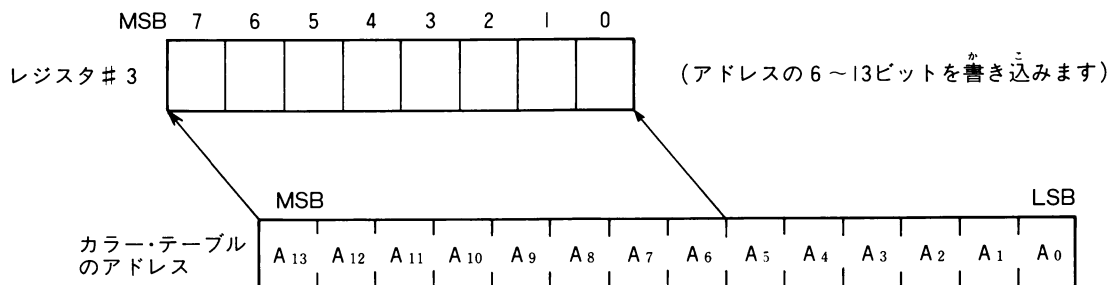
パターン名称テーブルの先頭アドレスを設定します。



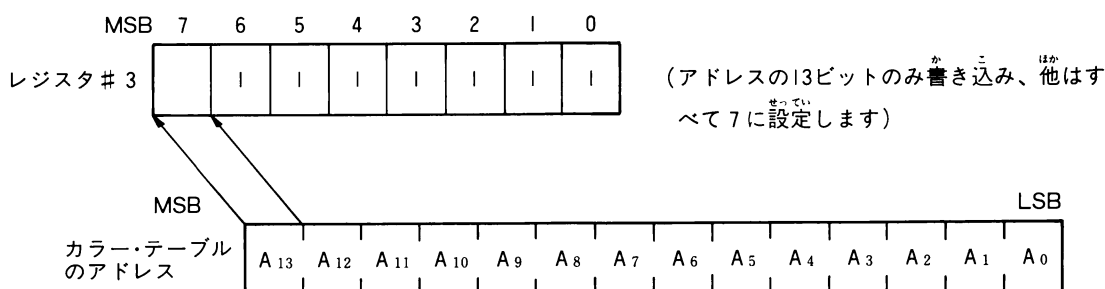
iii) レジスタ# 3

カラーテーブルの先頭アドレスを設定します。表示モードによって設定方法が異なります。

表示モード I の場合



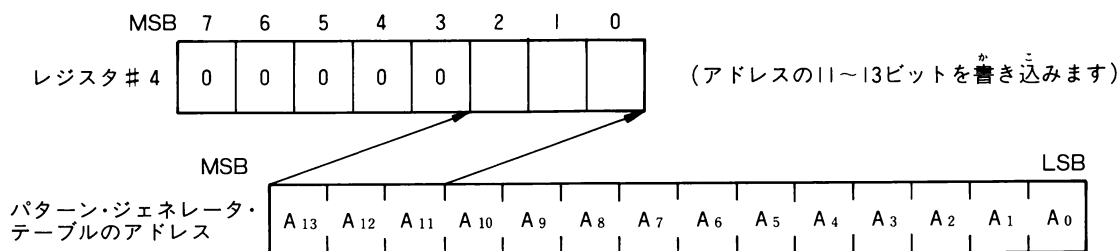
表示モード II の場合



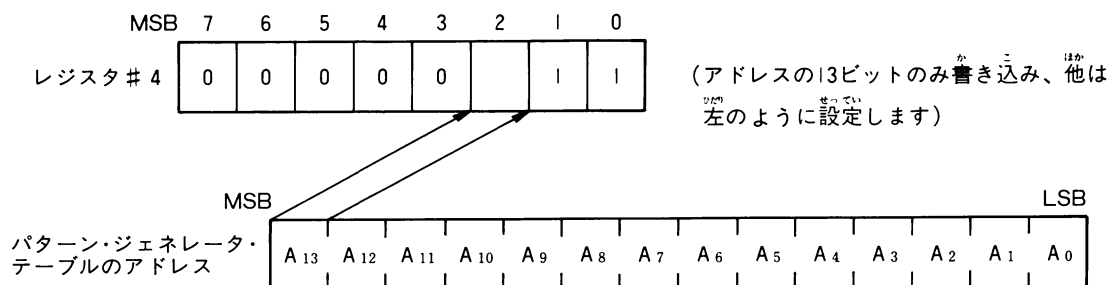
iv) レジスタ# 4

パターンジェネレータ・テーブルの先頭アドレスを設定します。表示モードによって設定方法が異なります。

表示モード I の場合

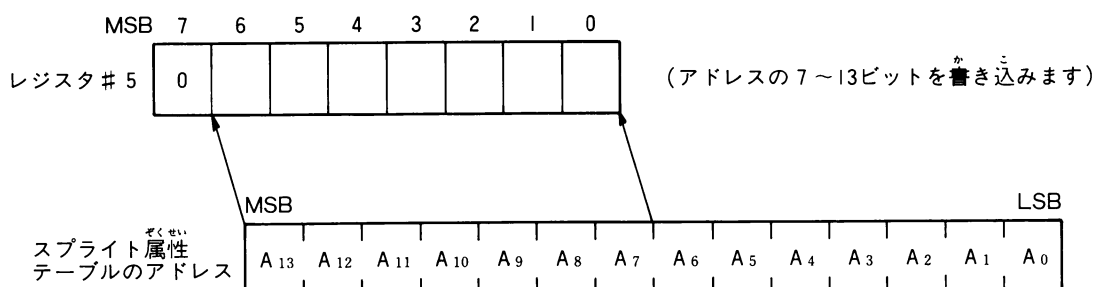


表示モード II の場合



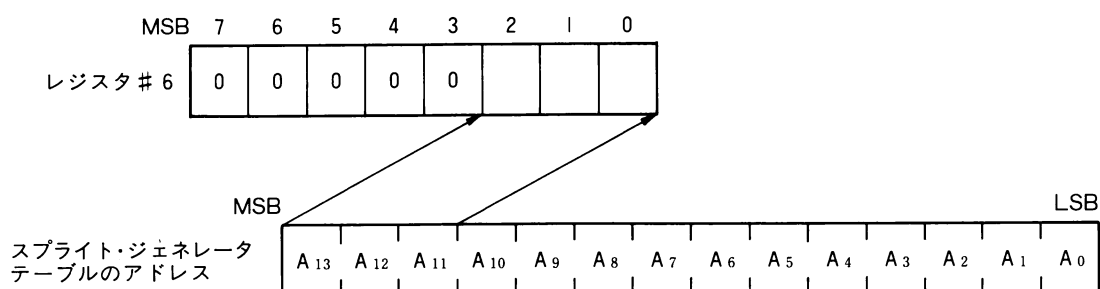
v) レジスタ#5

スプライト属性テーブルの先頭アドレスを設定します。



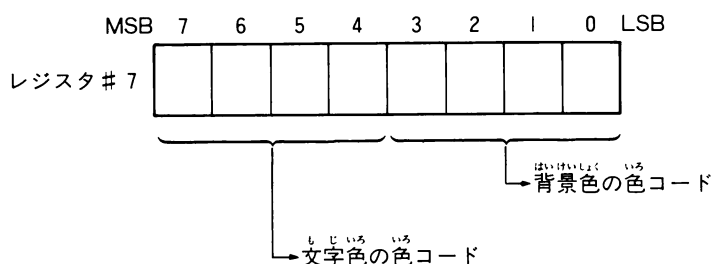
vi) レジスタ#6

スプライト・ジェネレータ・テーブルの先頭アドレスを設定します。



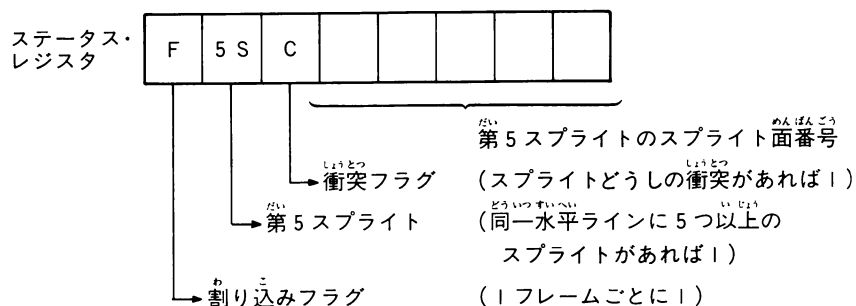
vii) レジスタ#7

文字色 (テキストモードの場合)、背景色を設定します。



viii) ステータス・レジスタ

VDP の動作状態を示します。



●VDP を実際に使用するにあたっては、V-RAM のマッピング、各テーブルの意味、表示法などについて、高度な理解を必要とします。

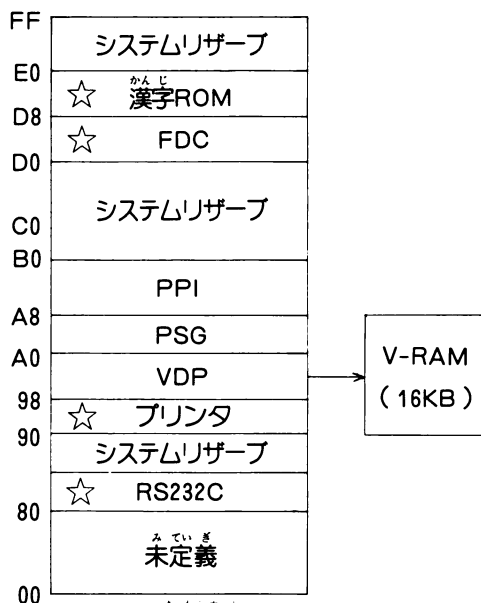
参考図書に示すような、もっと詳細なマニュアルを参照することをおすすめします。

(1)日本テキサスインスツルメンツ(株)、ビデオ・ディスプレイ・プロセッサ システム・デザイン・ハンドブック、共立出版、1983年

(2)TMS9918A の機能と使い方、月刊トランジスタ技術、1983年10月号、P.264～

アドレスマップ

●I/Oマップ

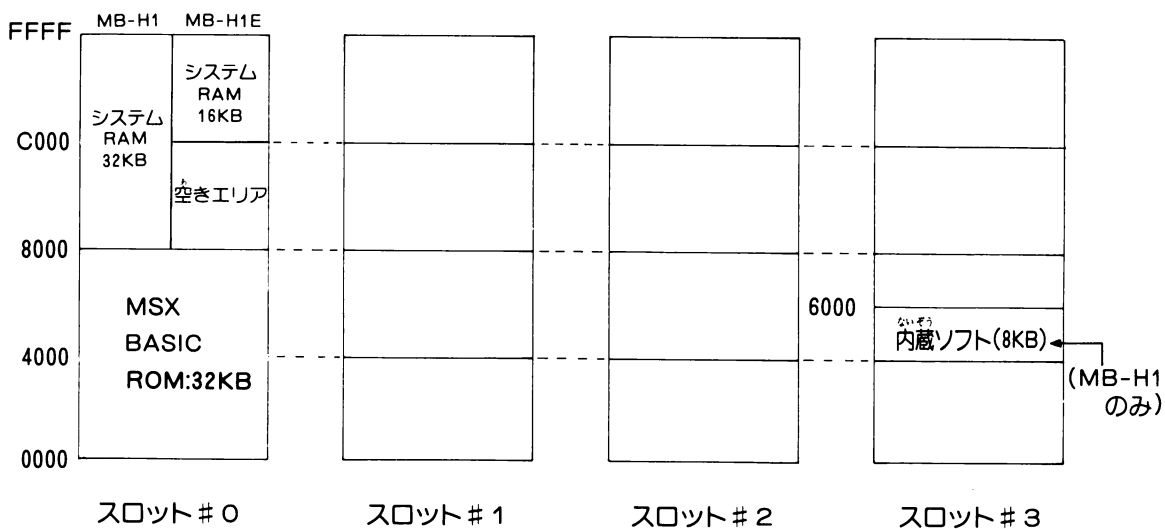


IO ADR	RW	内容	備考
&H98	W	V-RAMへのタータライト	9918A 相当品 (VDP)
&H99	R	V-RAMからのタータリロード	
	W	コマンド、アドレスセット	
	R	ステータスリロード	
&HA0	W	アドレスラッチ	AY-3-8910 相当品 (PSG)
&HA1	W	タータライト	
&HA2	R	タータリロード	8255A 相当品 (PPI)
&HA8	W	ポートAタータライト	
	R	リード	
&HA9	W	ポートBタータライト	
	R	リード	
&HAA	W	ポートCタータライト	
	R	リード	
&HAB	W	モードセット	
&H90	W	ストローブ出力 (b0)	ラッチ出力 BUSY '1' ラッチ出力 (プリンタ)
	R	ステータス入力 (b1)	
&H91	W	プリントデータ	

W : 書き込み用 R : 読み出し用

☆ MSX規格推奨のI/Oマップです。

●メモリマップ



ユーザ開放 (かいほう)

番地はすべて、16進数で表記されています。

コントロールコード一覧表

【CTRL】キーを押しながら以下のキーを押します (DELは除く)。

キー	コード		同じ機能 のキー	機能
	10 進	16 進		
B *	2	0 2		カーソルを1つ前の語の先頭に移動 (語の途中で押した場合はその語の先頭に移動)
C *	3	0 3		入力待ちの状態を中断する
E *	5	0 5		その行のカーソル以後を消去
F *	6	0 6		カーソルを次の語の先頭に移動
G *	7	0 7		ブザーをならす
H	8	0 8	BS	カーソルを1文字戻し、そこにあった文字を消去
I	9	0 9	TAB	次のタブストップまでカーソルを移動
J *	10	0 A		ライン・フィード(次の行にカーソルを移動)
K *	11	0 B	HOME	カーソルを画面左上に移動
L *	12	0 C	CLS	全画面を消去してカーソルを左上すみに移動
M *	13	0 D	RETURN	キャリッジ・リターン(1行の終了)
R *	18	12	INS	インサート・モードの切り換え
U *	21	15		その1行を消去
X *	24	18	SELECT	
¥ *	28	1 C	⇒	カーソルを右に移動
]	29	1 D	⇐	カーソルを左に移動
^ *	30	1 E	↑	カーソルを上移動
_ *	31	1 F	↓	カーソルを下移動
DEL	127	7 F	DEL	カーソル位置の文字を消去します

注) 「*」マークのついたコントロール・コードを入力すると、インサート・モードは通常モードに復帰します。

注) CLSキーはSHIFTキーを押しながら入力します。

キャラクターコード一覧表 カラーコード一覧表

●キャラクターコード一覧表

2バイトコード

(上位1バイト01H)

1バイトコード

下位4ビット		上位4ビット																	
		4	5																
0			π																
1	月																		
2	火																		
3	水																		
4	木																		
5	金																		
6	土																		
7	日																		
8	年																		
9	円																		
10	時																		
11	分																		
12	秒																		
13	百大																		
14	千中																		
15	万小																		

上位4ビット															
2	3	4	5	6	7	8	9	10	11	12	13	14	15		
	0	@	P	'	p	♠			ー	タ	ミ	た	み		
!	I	A	Q	a	q	♥	あ	。	ア	チ	ム	ち	む		
"	2	B	R	b	r	♣	い	「	イ	ツ	メ	つ	め		
#	3	C	S	c	s	♦	う	」	ウ	テ	モ	て	も		
\$	4	D	T	d	t	○	え	、	エ	ト	ヤ	と	や		
%	5	E	U	e	u	●	お	・	オ	ナ	ユ	な	ゆ		
&	6	F	V	f	v	を	か	ヲ	カ	ニ	ヨ	に	よ		
・	7	G	W	g	w	あ	き	ア	キ	ヌ	ラ	ぬ	ら		
(8	H	X	h	x	い	く	イ	ク	ネ	リ	ね	り		
)	9	I	Y	i	y	う	け	ウ	ケ	ノ	ル	の	る		
*	:	J	Z	j	z	え	こ	エ	コ	ハ	レ	は	れ		
+	:	K	[k		お	さ	オ	サ	ヒ	ロ	ひ	ろ		
,	<	L	¥	l		や	し	ヤ	シ	フ	ワ	ふ	わ		
ー	=	M]	m		ゆ	す	ユ	ス	ヘ	ン	へ	ん		
・	>	N	^	n	~	よ	せ	ヨ	セ	ホ	ゝ	ほ			
/	?	O	—	o		つ	そ	ツ	ソ	マ	°	ま	■		

カーソル

●カラーコード一覧表

カラーコード	色	カラーコード	色
0	透明	8	赤
1	黒	9	明るい赤
2	緑	10	暗い黄
3	明るい緑	11	明るい黄
4	暗い青	12	暗い緑
5	明るい青	13	紫
6	暗い赤	14	灰
7	水色	15	白

さん こう と しよ 参考図書

とうしゃ 当社のパーソナルコンピュータ、MSX-BASIC などについて次の表のような図書が市販されています。

しやうわ ねん がつげんざい
昭和58年 11 月現在

と しよ めい 図 書 名		しゅつ ぼん しや 出 版 社	おも な い 主 な 内 容
月 刊	アイ オー I/O	こうがくしや 工学社	ベーシック、機械語プログラムを 使用したゲームなど豊富な 内容 (入門者から上級者まで)
	マイコン	でん ぱ しん ぶん しや 電波新聞社	
	アスキー ASCII	アスキー	
	ラ ム RAM	こう さい どうしゅ ばん 廣済堂出版	
誌	インターフェース	シーキュー しゅつ ばん CQ出版	やや高度な内容で上級者向き
	がくしゅ 学習コンピュータ	がっ けん 学 研	マイコンから高級言語のプログラ ミングまで(初級～中級向き)
	エムエスエックス MSX マガジン	アスキー	MSX ベーシックを中心にした ゲームなど豊富な内容

じやう き い がい 上記以外にも、パーソナルコンピュータ関係の雑誌、単行本など、多数出版されておりますので参考にすることをおすすめします。

また、MSX-BASICについては、今後、多数の出版物が予想されますので、参考にすることをおすすめします。

エラーメッセージ一覧表

エラーコード	エラーメッセージ	内 容
1	テキスト ウィズアウト フォー NEXT without FOR	FOR～NEXTが正しく対応していない。(FOR文がないのにNEXT文に出会った。)
2	シンタックス エラー Syntax error	プログラムが文法にそっていない。(打ち間違いや書き方の間違いがある。)
3	リターン ウィズアウト ゴーサブ RETURN without GOSUB	GOSUB～RETURNが正しく対応していない。(GOSUB文がないのにRETURNに出会った。)
4	アウト オフ データ Out of DATA	READすべきDATA文がない。(データを読みつくした後、さらにREAD文がある。)
5	イリーガル ファンクション コール Illegal function call	関数やステートメントの機能の呼び方が間違っている。
6	オーバーフロー Overflow	計算結果や入力した数値がベーシックの数値形式の許容範囲をこえている。
7	アウト オフ メモリー Out of memory	メモリ容量が足りなくなった。(プログラムが長すぎる、ファイルが多すぎる、変数が多すぎるなど。)
8	アンディファインド ライン ナンバー Undefined line number	指定した行番号が存在しない。
9	サブスクリプト アウト オフ レンジ Subscript out of range	配列の添字が規定の範囲内にない。(添字の数値が間違っている。)
10	リディメンションド アレイ Redimensioned array	同じ配列を二重に定義している。または初期値の10の配列が成立した後で、さらに配列を定義している。
11	ディビジョン バイ ゼロ Division by zero	0で割り算をしています。
12	イリーガル ダイレクト Illegal direct	直接モードで使うことのできないコマンドを直接モードで実行しようとした。(DEFFN命令など)
13	タイプ ミスマッチ Type mismatch	数値と文字列というように、変数または定数の型、関数の引数の型があわない。

エラーコード	エラーメッセージ	内 容
14	アウト オブ スtring スペース Out of string space	文字変数用のメモリエリアがたりなくなった。
15	String too long	文字式の値の文字数が255を超えた。
16	String formula too complex	文字式が複雑すぎる。
17	Can't CONTINUE	Count 命令によるプログラムの続行ができない。(エラーにより実行が停止している、[CTRL] + [STOP] の後、プログラムの修正を行なった。または続行すべきプログラムがない。)
18	Undefined user function	DEF FNにより、まだ定義されていないユーザ関数が呼ばれた。
19	Device I/O error	カセットまたはプリンタ、ディスプレイに出力エラーが発生した。ベーシックでは、このエラーから回復することはできません。
20	Verify error	カセットにセーブしたプログラムが現在のプログラムとは異なっている。
21	No RESUME	エラー処理ルーチンにRESUMEがない。
22	RESUME without error	エラーとRESUME 文が対応していない。(エラーが発生していないのにRESUMEを実行しようとした。)
23	Unprintable error	エラーメッセージの定義されていないエラーが発生した。
24	Missing operand	演算に必要な数値が指定されていない。(式の右辺がないなど)
25	Line buffer overflow	入力された行の文字数が多すぎる。
26 49	Unprintable error	エラーの定義がされていませんが、ベーシック拡張の際使用されることが予約されています。

エラーコード	エラーメッセージ	内 容
50	フィールド オーバーフロー FIELD overflow	ランダムファイルの指定されたレコード長のバイト数を超えている、またはFIELDバッファがランダムファイルへのシーケンシャル I/O 実行中が終わってしまった。
51	インターナル エラー Internal error	ベーシック 内部に異常が発生した。通常発生することはありませんが、万一生じた場合には、電源を一度切って入れなおしてください。
52	バッド ファイル ナンバー Bad file number	OPENされていないファイルのファイル番号を使用した。またはMAXFILE文により指定したファイル番号の範囲を超えたファイル番号のファイルを使用した。
53	ファイル ノット ファウンド File not found	指定されたファイルが見つからない。
54	ファイル オールレディ オープン File already open	すでにOPENされているファイルをさらにOPENしようとした。
55	インプット パスト エンド Input past end	ファイルの全てのデータを読みつくした後に、Input命令を実行した。
56	バッド ファイル ネーム Bad file name	ファイル名やデバイスディスクリプタのつけ方を間違えた。
57	ダイレクト ステートメント イン ファイル Direct statement in file	ASCII 形式 ファイル中に、直接モードの命令文がある。
58	シーケンシャル アイ・オー オンリー Sequential I/O only	シーケンシャルファイルに対し、ランダムアクセス命令を実行した。
59	ファイル ノット オープン File not OPEN	OPENされていないファイルを使用しようとした。
60 } 255	アンプリンタブル エラー Unprintable error	未定義のエラーコードです。ユーザが任意にエラーコードを定義して使用することができます。



この画面は、MB-H1^{ないそう}内蔵プログラムの画面です。

MB-H1E^{ないそう}には、内蔵プログラムはありません。

日立家電販賣株式会社

〒105 東京都港区西新橋2丁目15番12号
電話(03)502-2111

株式会社日立製作所

〒105 東京都港区西新橋2丁目15番12号
電話(03)502-2111